

**NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
“IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE”
Educational and Research Institute for Applied System Analysis
Department of Artificial Intelligence**

The defense allowed:

Acting head of the department

_____ Iryna DZHYGYREY

« ____ » _____ 20__ p.

**Diploma work
for a bachelor's degree
in Educational and Professional Program
«Systems and Methods of Artificial Intelligence»
specialty 122 «Computer sciences»
on the topic: «Development of a personalized content recommendation system
for streaming platform based on user behavior analysis »**

Completed:

IV year student, group KI-01

Aliiev Ramin Nazirovich _____

Supervisor:

Associate professor, Ph. D.

Huskova V.H. _____

Adviser for the economic section:

Department of EC FMM, Prof. Dr.

Olena Anatoliivna Shevchuk, _____

Adviser for the control of norms:

the first category specialist of the Department of AI,

Kravets P.V. _____

Reviewer:

Doctor of technical sciences, Professor

Dmitrieva Olga Anatoliivna _____

Hereby I certify that this diploma work
does not contain any borrowed material
from the works of other authors without
appropriate references.

Student _____

Kyiv – 2024 year

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«__» _____ 20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

**на тему: «Розробка персоналізованої системи рекомендацій контенту для
поточної платформи на основі аналізу поведінки користувача»**

Виконав:

студент IV курсу, групи КІ-01
Алієв Рамін Назірович

Керівник:

Асистент, Ph. D.

Гуськова Віра Геннадіївна _____

Консультант з економічного розділу:

професор, д.е.н., проф. кафедри ЕК ФММ,
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:

фахівець першої категорії кафедри ІІІ,
Кравець П.В. _____

Рецензент:

Д.т.н., професор

Дмитрієва Ольга Анатоліївна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студент _____

Київ – 2024 року

National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
Educational and Research Institute for Applied System Analysis
Department of Artificial Intelligence

Level of higher education – first (bachelor)

Specialty (program subject area) – 122 «Computer sciences»

Educational and Professional Program –
«Systems and Methods of Artificial Intelligence»

APPROVED

Acting head of the department

_____ Iryna DZHYGYREY

31st January 2024

ASSIGNMENT
for the student's diploma work

Aliiev Ramin Nazirovich

1. Topic of the project «Development of a personalized content recommendation system for streaming platform based on user behavior analysis», work's supervisor Huskova. V. H., Ph.D., Associate professor approved by the University Order of «31» May 2024 №2240-C
2. Deadline for submission of the project: 10th June 2024.
3. Initial data for the project.
4. Content of the work.

5. List of the graphic material (indicating mandatory drawings, posters, presentations, etc.)

6. Advisers for sections of the work

Section	Surname, initials and position of the adviser	Signature, date	
		task issued	task accepted
Economic	Department of EC FMM, Prof. Dr. Olena Anatoliivna Shevchuk,		

7. Date of the task issue: 5th February 2024.

Calendar plan

№ by number	The deadline for completing the stages of the diploma work	Deadline	Notes
1	Literature review on the topic	25.03.2024	Completed
2	Preparation of the first chapter	10.04.2024	Completed
3	Preparation of the second chapter	20.04.2024	Completed
4	Development of a software product	15.05.2024	Completed
5	Preparation of the third section	22.05.2024	Completed
6	Preparation of the economic part	28.05.2024	Completed
7	Preparation of the diploma work	08.06.2024	Completed
8	Preparation of the presentation of the report	13.06.2024	Completed
9	Literature review on the topic	25.03.2024	Completed

Student

Ramin ALIIEV

Supervisor

Vira HUSKOVA

ABSTRACT

Thesis: 101 pages, 18 tables, 26 figures, 1 appendix, 27 sources.

RECOMMENDATION SYSTEM, MACHINE LEARNING, COLLABORATIVE FILTERING, STREAMING PLATFORMS

The object of research is the process of creating a recommendation system for selecting movies on a streaming platform.

The subject of research is the methods and algorithms used to create content personalization systems.

The aim of the work is to investigate methods and algorithms for building recommendations and increasing their efficiency, and to integrate the selected algorithm into a movie recommendation application.

The relevance of this work is associated with the need to predict user preferences based on data collected from their interactions and feedback.

The research involves analyzing the problem of developing a recommendation system and predicting user preferences, justifying the use of specific methods, tools, and development environments, preparing and analyzing the initial dataset, preprocessing raw data, developing mathematical models for the recommendation problem, implementing the solution, and analyzing the results obtained.

Experiments and comparisons with existing approaches are conducted to evaluate the effectiveness of the developed recommendation system. The results of the experiments demonstrate the advantages of the developed movie recommendation system for the streaming platform.

РЕФЕРАТ

Дипломна робота: 101 с., 18 табл., 26 рис., 1 додаток, 27 джерел.

РЕКОМЕНДАЦІЙНА СИСТЕМА, МАШИННЕ НАВЧАННЯ,
КОЛАБОРАЦІЙНА ФІЛЬТРАЦІЯ, СТРИМІНГОВІ ПЛАТФОРМИ

Об'єкт дослідження – процес створення рекомендаційної системи для вибору фільмів на стрімінговій платформі.

Предмет дослідження – методи та алгоритми, які використовуються для створення систем персоналізації контенту.

Мета роботи – дослідження методів та алгоритмів для побудови рекомендацій та підвищення їх ефективності, а також подальша інтеграція обраного алгоритму в додаток для рекомендацій фільмів.

Актуальність роботи пов'язана з необхідністю прогнозування уподобань користувачів на основі даних, зібраних з їх взаємодій та відгуків.

У дослідженні аналізується проблема розробки рекомендаційної системи та прогнозування уподобань користувачів, обґрунтовується використання конкретних методів, інструментів та середовищ розробки, проводиться підготовка та аналіз початкового набору даних, попередня обробка сирих даних, розробка математичних моделей для задачі рекомендації, реалізація рішення та аналіз отриманих результатів.

Для оцінки ефективності розробленої рекомендаційної системи проводяться експерименти та порівняння з існуючими підходами. Результати експериментів демонструють переваги розробленої системи рекомендацій для фільмів на стрімінговій платформі.

CONTENTS

INTRODUCTION.....	11
1. OVERVIEW OF RECOMMENDER SYSTEMS	13
1.1 History and evolution of recommender systems.....	13
1.2 Importance in Streaming Platforms	14
1.2.1 Case studies of popular streaming platforms.....	14
1.2.2 Example of recommendation system components	16
1.3 Statement of the research problem	19
Conclusions to section 1	20
2 ANALYSIS OF THE FEATURES OF THE CONSTRUCTION OF RECOMMENDATIONS.....	21
2.1 Recommendation Approaches	21
2.1.1 Content-Based Filtering.....	22
2.1.2 Collaborative-Based Filtering.....	24
2.1.3 Hybrid Filtering	27
2.1.4 Knowledge-based Systems	30
2.1.5 Demographic Systems	30
2.1.6 Community-based Systems.....	31
2.2 Challenges in Recommendation systems.....	32
2.2.1 Cold-start Problem	32
2.2.2 Sparsity	33
2.2.3 Scalability	34
2.2.4 Shilling attacks.....	34
2.2.5 GraySheep Problem	35

	8
2.3 Data Collection	35
2.3.1 Explicit Data	36
2.3.2 Implicit Data	37
2.4 Recommendation Algorithms	37
2.4.1 Computing User Similarity	37
2.4.2 Computing Item Similarity	39
2.4.3 For Neighbourhood Generation	40
2.5 Metrics for Evaluating Recommender Systems	41
2.5.1 Coverage	41
2.5.2 Precision.....	41
2.5.3 Recall	42
2.5.4 Hit Rate	42
2.5.5 Leave-One-Out Cross-Validation.....	43
2.6 Learning Types	44
2.6.1 Unsupervised.....	44
2.6.2 Supervised.....	44
Conclusions to section 2	45
3 PRODUCT DEVELOPMENT AND EVALUATION PROCESS.....	46
3.1 Development Process Overview.....	46
3.2 Choosing programming languages, frameworks and libraries for implementation.....	47
3.2.1 Frontend	47
3.2.2 Backend.....	48
3.2.3 Recommender engine	48
3.2.4 Environment.....	50

3.2 Data collecting	50
3.2.1 Datasets	50
3.2.2 Data Preparation for Collaborative Filtering with Apache Spark ...	53
3.3 Explanation of the choice of algorithm and its implementation.....	55
3.4 Evaluation of ALS Algorithm for Recommendation System.....	59
3.4.1 Evaluation Metrics	59
3.4.2 Methodology	59
3.4.3 Experimental Setup.....	60
3.4.4 Results and Analysis.....	60
3.5 Integration of User Interface with Backend Recommendation Engine..	62
3.5.1 Frontend Development	62
3.5.2 Backend.....	63
Conclusions to section 3	64
4 FUNCTIONAL AND COST ANALYSIS OF A SOFTWARE PRODUCT	66
4.1 Formulation of the Design Problem	66
4.2 Justification of the Functions of the Software Product	67
4.3 Justification of the software product parameter system	69
4.4 Analysing the expert evaluation of parameters	72
4.5 Analysis of the quality level of options for implementing functions	76
4.6 Economic analysis of the options for developing the PP.....	78
4.7 Selection of the best option for the PP at the technical and economic level	84
Conclusions to section 4	85
SUMMARY	86

REFERENCES.....	87
APPENDIX A – PROGRAMCODE	90

INTRODUCTION

The exponential growth of such popular streaming platforms as Netflix, Spotify, and YouTube has extremely transformed how people are consuming media. Each user of a streaming platform is looking for the content that matches their interests and preferences. However, given the huge amount and diversity of content available, this can be a quite challenging task. On the one hand, some users are willing to discover new and interesting content without the frustration of endless searching. On the other hand, streaming platforms aim to enhance user satisfaction and engagement by providing relevant recommendations. Thanks to the development of artificial intelligence technologies and machine learning, the possibility of creating intelligent recommender systems capable of analyzing user behavior, their preferences and needs, offering them relevant goods and services. This is where modern machine learning, deep learning, and natural language processing including collaborative filtering (CF), content-based filtering, and user-based collaborative filtering methods, come into play.

The relevance of this work stems from the need to predict user preferences for content on streaming platforms based on data collected from user interactions and feedback.

The goal of this work is to solidify knowledge, skills, and to deep into the application of data mining methods to solve the problem of content recommendation based on the available dataset, considering the user's behaviour and specifics of the task.

The tasks of the work include analyzing the problem of development a recommender system and predicting user preferences, justifying the use of specific methods, tools, development environments, preparing and analyzing the initial dataset, preprocessing raw data, developing mathematical models for the recommendation problem, implementing the solution, and analyzing the results obtained.

The structure of the thesis consists of Introduction, Overview of modern recommendation systems, Theoretical background, Development process, Final evaluation, Conclusions, References and Finally appendices. Overview provides a history and the evolution of recommendation systems throughout the time. Theoretical background sets the foundation for the development process, diving into the types of recommending algorithms, different learning approaches, and evaluation metrics. The development process chapter describes the methodology, including the dataset, preprocessing steps, and the specific techniques and algorithms used to develop the recommendation system. Final evaluation presents experimental results, and performance analysis, evaluating the effectiveness of the developed system.

1. OVERVIEW OF RECOMMENDER SYSTEMS

Below, a brief introduction and history perspective to recommender systems will be given, which problems they address and the benefits they provide.

1.1 History and evolution of recommender systems

In the mid-1970s, recommendation systems emerged as a distinct area of research at Duke University. Since then, numerous studies have been conducted, leading to the development of various approaches. The first known recommender system was Tapestry, created at the Xerox Palo Alto Research Center. Tapestry was developed to manage the increasing influx of emails, many of which were unnecessary and difficult to handle [1]. To address this issue, users could create mailing lists that allowed only certain contacts to send them emails, similar to modern email filtering systems. Tapestry enabled searches based on document content and user feedback, using collaborative filtering to recommend documents. For example, a user could request all documents containing "racing bike" that another user, "William," rated as "excellent."

Following Tapestry, recommender systems gained popularity and became integral to various internet platforms, such as Amazon, Pandora, Netflix, dating and social networking sites, YouTube, and Tripadvisor. By the late 1990s, the industry began investing heavily in recommendation system research. One notable example is the Netflix Prize, launched in 2006, which offered \$1 million to the team that could develop the best movie recommendation system. The winning team was announced in 2009. In 2010, YouTube also implemented a recommendation system on its platform.

The development of recommendation systems has been closely linked with advancements in artificial intelligence, information retrieval, and human-computer interaction, which have enhanced their efficiency and popularity [2].

1.2 Importance in Streaming Platforms

1.2.1 Case studies of popular streaming platforms

Artificial intelligence (AI) and machine learning are increasingly common buzzwords in online articles. Major websites across various industries use these technologies to gain a strategic advantage over competitors and maintain their market positions. AI-based solutions can significantly enhance the user experience on a website, leading to increased engagement, better click-through rates (CTR), and higher sales for the site owners. In this chapter, we look at the machine learning-based recommendation systems of the largest streaming platforms.

Netflix is a trendsetter in modern recommendation systems, that implemented the first model of its recommendation system in 2000. In 2006, Netflix introduced the "Netflix Prize," a competition offering \$1 million to the team that could develop the best data filtering algorithm. Today, Netflix employs an advanced recommendation engine utilizing four sophisticated algorithms. These algorithms consider factors such as movie descriptions, target age ranges, user ratings, and popularity. The recommendations are organized into multiple rows, sorted by categories and user preferences. Netflix reports that approximately 80% of its total watch time is driven by its recommendation system, highlighting its effectiveness. Additionally, the platform continuously enhances user experience to improve retention rates, leading to significant savings in customer acquisition costs—around \$1 billion annually since 2016 [3].

Spotify users are often impressed by the accuracy of the "Discover Weekly" playlist, which matches their music tastes. Every Monday, the platform generates 30

new song suggestions for millions of users worldwide. These playlists are created entirely by an automated recommendation engine powered by artificial intelligence algorithms. Most of the personalized content on Spotify is generated by the recommender system known as "Bandits for Recommendations as Treatments" (BaRT). This system includes three main algorithms that are usually used in other recommendation systems: collaborative filtering, natural language processing, and audio analysis. These algorithms work together to analyze song descriptions, audio features, and user behavior to create and suggest the most relevant recommendations. Given that Spotify users have listened to over 2.3 billion hours of music from "Discover Weekly" playlists since their inception, it is clear that these algorithms are highly effective in recommending new music [4]. For more insights into the secret behind Spotify's recommendation engine success, check out our blog.

TikTok is a relatively new social networking platform that allows users to upload creative videos up to 60 seconds long. Two key features distinguish TikTok from other platforms: a significantly higher engagement rate and one of the most advanced recommendation systems globally. Users do not need to search or decide in advance what they want to watch; the personalized feed delivers content likely to capture their interest with a single click. The algorithm analyzes real-time data about the videos, as well as preferences and data from other users, including viewing times. TikTok users typically spend 52 minutes per day on the app, whereas Snapchat, Instagram, and Facebook users spend only 26 minutes, 29 minutes, and 37 minutes respectively. This demonstrates the substantial advantage offered by AI-based solutions [5].

Since May 2019, over 500 hours of videos are uploaded to YouTube every minute. With millions of videos ranging from short, fun clips of animals to specialized tutorials, an effective recommendation system is crucial for helping users find what they are looking for. YouTube's recommendations are powered by Google Brain, which has recently been deployed as TensorFlow. This transition has significantly simplified the process for Google to train, test, and implement AI-based solutions across various environments. YouTube's recommendation engine comprises

two main neural networks. The first network is responsible for generating candidates by analyzing millions of videos and narrowing them down to a few hundred based on user behavior and other data, such as viewing history and search queries. The second neural network manages the ranking system. It takes into account data from the candidate generator and other available information, such as the user's language, the video's language, and the user's personal preferences, to rank the videos based on the likelihood of user interest. These recommendations are presented to users in real-time and are highly effective.

1.2.2 Example of recommendation system components

Recommendation blocks create the foundation for modern streaming platforms, playing the main role in enhancing user experience by offering content configured to individual preferences. Netflix serves as a primary example of the effectiveness and sophistication of current systems. Launched in 2000, Netflix's recommendation system has evolved outstandingly, due to continuous innovation and a focus on personalization of each user.

The platform employs a series of recommendation blocks and components, which are designed to target various aspects of user behavior and preferences. These blocks are powered by advanced machine learning algorithms and artificial intelligence, which analyze vast amounts of data, including explicit feedback like user ratings as well as implicit feedback like viewing history, clicks and demographic information. By organizing content into different recommendation blocks, such as "Trending Now," "Top Picks for You," and "Because You Watched," Netflix ensures that users are suggested with relevant and engaging content automatically with minimal effort.

The success of Netflix's recommendation system once again confirmed the fact that a significant part of the content (approximately 80%) of viewed content

is managed by these personalized suggestion blocks, what highlighted the importance of effective recommendation components. This part of the chapter will show us successful examples of implementing recommendation systems into streaming platforms.

Block **“Top 10 Movies/TV Shows in [Country] Today”** is created to provide a weekly Top 10 list of the most-watched TV and films, allowing users to see what others in their area are enjoying. This can be especially useful for discovering trending content that is culturally or regionally relevant.

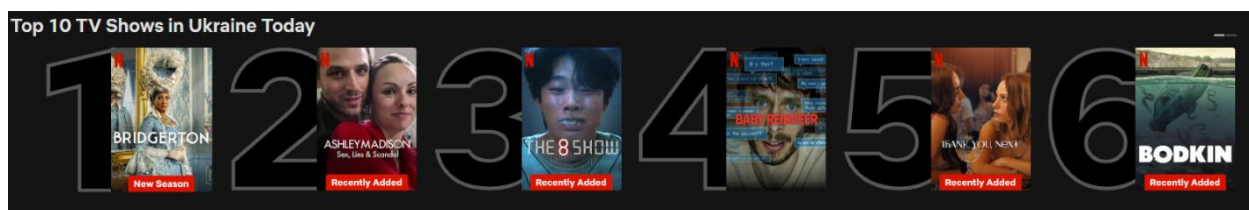


Figure 1.1 – Recommendation block “Top 10 TV Shows in Ukraine Today”

The "Top 10 Movies" block remarkably enhances user engagement using regional trends and preferences. Users are more likely to choose the movie that is popular among their peers, which in turn is boosting overall viewership for these titles. For content providers, this feature can stimulate localized marketing offers and efforts, and helps to adapt future content offerings to regional tastes. Additionally, it promotes a sense of community among users, as they can see and discuss the movies that are currently popular in their country.

To give a sense of what people are watching around the world, Netflix also publishes Top 10 lists for nearly 100 countries and territories (the same locations where there are Top 10 rows on Netflix). Country lists are also ranked by views.

Block **“Today's Top Picks for You”** uses advanced algorithms to analyze a vast amount of data like user's viewing history, ratings, behavior patterns, other members with similar tastes and preferences on our service. In addition, it considers time of day user watches, the devices user is watching Netflix on, and how long user

watches. By considering factors such as genres, actors, directors, and themes that a user tends to enjoy, the system generates a list of recommendations that are highly likely to appeal to them.

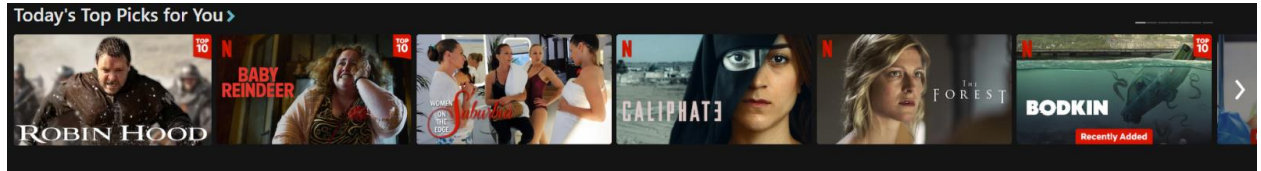


Figure 1.2 – Recommendation block “Today's Top Picks for You”

For streaming services, it helps to retain users by constancy of offering relevant and engagement content, which can reduce churn rates and improve user’s loyalty. Additionally, it allows for more efficient content discovery, ensuring that users make the most of their subscription and stay engaged with the platform.

Block "**New on Netflix**" has a main focus on the most recent releases, what ensures that users are becoming the latest content added to the platform. This block is updated constantly to include new arrivals, what promotes helping users to stay informed about the newest added movies or TV Shows to Netflix's extensive catalog. It usually can include a mix of different genres and formats, catering to a wide range of tastes and preferences.

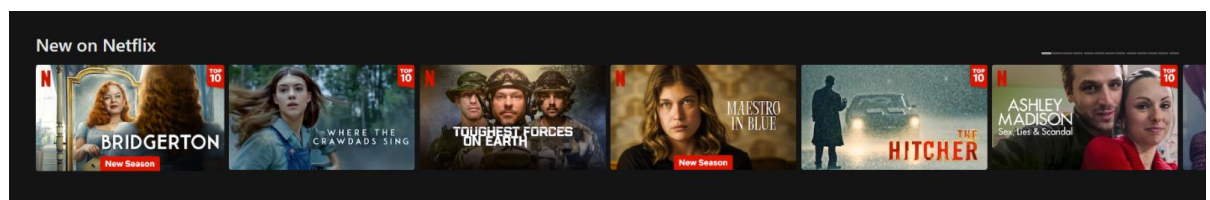


Figure 1.3 – Recommendation block “New on Netflix”

By creating top lists of new releases, it contributes to encouraging users to explore and watch the recently added movies and shows, what promotes viewership for newly added titles. This feature also helps to attract and retain subscribers who are

looking for the most up-to-date content. For content creators and distributors, showing their content in this section can provide immediate visibility and can lead to higher initial viewership numbers. In addition, it stimulates a sense of anticipation and excitement among users, as they are looking forward to discover new content on the regular basis.

1.3 Statement of the research problem

The object of research in the bachelor's thesis is the process of creating a recommendation system for choosing a movie on a streaming platform. The subject of research is the methods and algorithms used to create such systems of content personalization.

The purpose of the study is to investigate methods and algorithms for building recommendations and increase their efficiency and further integration of the selected algorithm into the recommendation app for movies. In order to achieve these goals, it is necessary to perform the following tasks:

- conduct a study of the subject area and available data;
- compare existing methods of building recommendation systems and choose the one that is most suitable and meets the needs and solves the tasks;
- modify the existing selected method to meet the needs of the site;
- develop a software implementation of the recommendation system based on a previously created model, using an up-to-date library for the Python programming language;
- evaluate the quality of the resulting model using current metrics for recommendation systems.

The research should result in the development of a method for creating recommender systems that outperforms previous approaches in terms of efficiency, as

well as software that will implement this system concept recommendations for movie selection on the streaming platform.

Conclusions to section 1

In this section, we explored the history and evolution of recommender systems, tracing their development from the mid-1970s at Duke University to their current widespread use in various digital platforms. We examined the early example of Tapestry, which pioneered collaborative filtering to manage email overload, and highlighted the significant advancements and applications of recommendation systems in major platforms like Amazon, Netflix, Spotify, YouTube, and TikTok. We also discussed how these systems integrate artificial intelligence and machine learning to enhance user experience and engagement effectively.

2 ANALYSIS OF THE FEATURES OF THE CONSTRUCTION OF RECOMMENDATIONS

The development of recommendation systems involves a different methods and approaches, and each is created to solve a specific problem and to use different types of data. This chapter dives into the theoretical foundations of these systems, exploring their development, applications, pros and cons. We will discuss three main types of recommender systems – content-based filtering, collaborative filtering, and hybrid filtering – going into the details of each methodology. Additionally, we will discuss data collection techniques, both explicit and implicit feedback, and different approaches to learn, including supervised and unsupervised learning. Finally, we will cover metrics to evaluate the performance and accuracy of recommender systems, what provides a complex understanding of this technology.

2.1 Recommendation Approaches

To gain a deeper understanding of recommender systems, the first section outlines the most prevalent recommendation approaches: collaborative filtering, content-based filtering, hybrid recommender systems, and knowledge-based systems. Additionally, it explores other recommendation methods, such as demographic-based systems and community-based systems. The subcategories within these primary approaches are also briefly discussed.

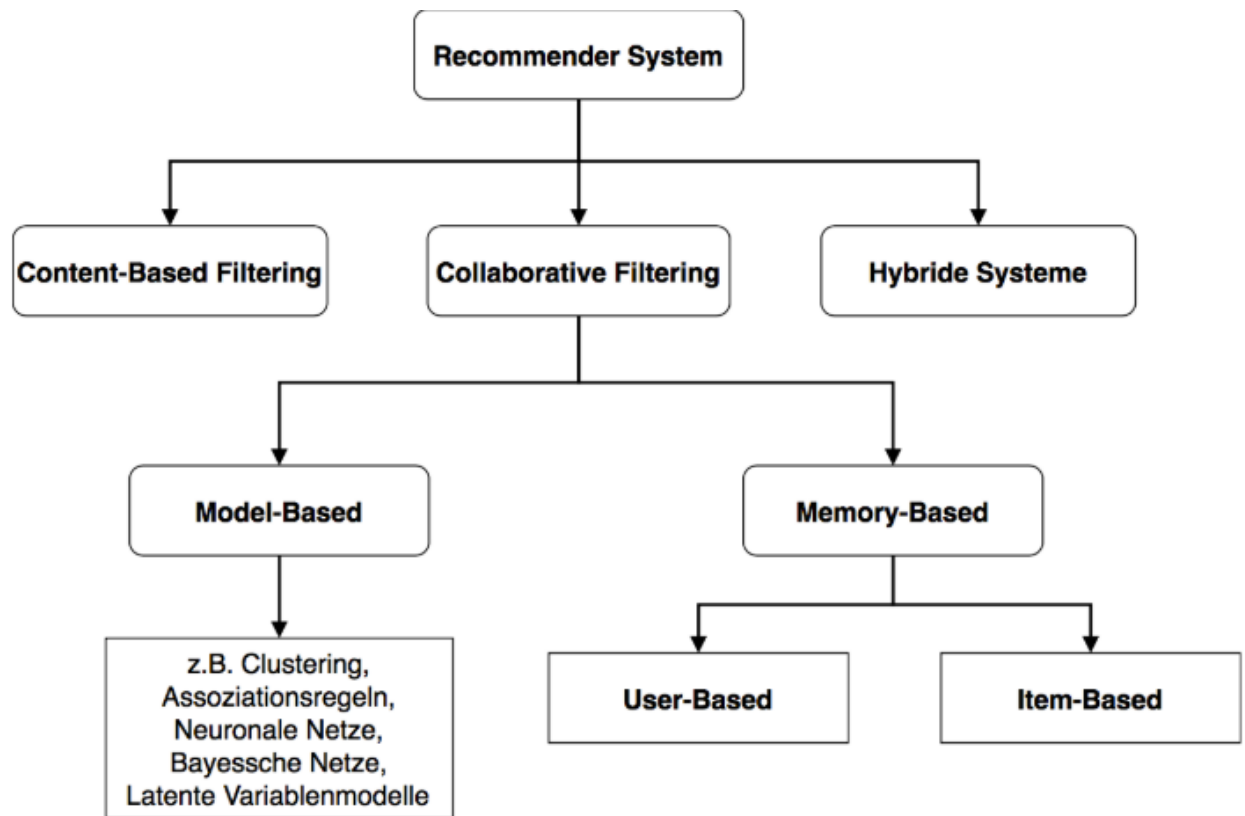


Figure 2.1 – The main types of recommendation system [7].

2.1.1 Content-Based Filtering

Content-based filtering is one of the two primary types of recommender systems, recommending items to users based on the features of individual items. This method is an information retrieval technique that selects and returns items relevant to a user's query by considering the features of other items the user has shown interest in. Although the term "content-based" might suggest otherwise, some recommendation algorithms match items according to descriptive features (such as metadata) attached to items rather than the actual content. However, several content-based methods, like content-based image retrieval or natural language processing applications, do match items based on their intrinsic attributes [10].

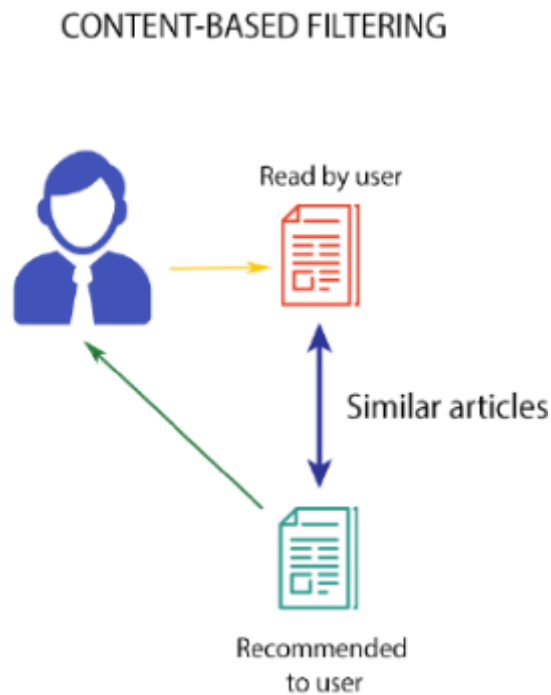


Figure 2.2 – Content-Based Filtering [8].

Component Procedures of a Recommender:

In content-based recommender systems (CBRSs), the recommendation engine compares a user profile with an item profile to predict user-item interactions and suggest items accordingly. The item profile represents the item within the system, consisting of a feature set that includes either internal structured characteristics or descriptive metadata. For example, a streaming service might categorize movies by genre, release date, director, and other attributes.

The user profile captures user preferences and behavior, comprising representations of items the user has previously shown interest in. It also includes data from past interactions with the system, such as likes, dislikes, ratings, and queries [9].

Example of usage CBF

A content-based recommender system attempts to predict a user's preferences or behavior based on the features of items they have reacted positively to.

Table 2.1 – User preference table [10]

Movies	User 1	User 2	User 3	User 4	Action	Comedy
Item 1	1		4	5	Yes	No
Item 2	5	4	1	2	No	Yes
Item 3	3	4	2		No	No
Item 4	4	2	2	4	Yes	Yes

The last two columns, Action and Comedy, describe the genres of the movies. By analyzing these genres, we can identify which users prefer which genres and subsequently determine the features corresponding to each user based on their reactions to movies of those genres.

Once we understand a user's preferences, we can represent them in an embedding space using a generated feature vector and make recommendations based on their choices. During the recommendation process, similarity metrics (which we will discuss shortly) are calculated between the item's feature vectors and the user's preferred feature vectors from their past interactions. The top recommendations are then selected based on these similarities.

Content-based filtering does not require data from other users when making recommendations for a single user.

2.1.2 Collaborative-Based Filtering

Collaborative filtering does not require the features of the items to be provided explicitly. Instead, it generates feature vectors or embeddings for both users and items on its own, embedding them in the same space. This approach considers other users' reactions while making recommendations for a particular user. It observes which items a user likes and also considers the items liked by other users with similar behavior and preferences. By collecting user feedback on various items, collaborative filtering uses this information to make personalized recommendations [10].



Figure 2.3 – Collaborative-Based Filtering [8].

Memory-Based Filtering

Memory-based approaches use all or part of the user item database. User rating data is used to calculate similarities between users and items. The best-known techniques are algorithms based on neighbourhood methods. Recommendation generation can be divided into two steps: Calculating the similarities between users and items and making predictions for unknown ratings. A distinction is made between two approaches in memory-based filtering [2].

User-based collaborative filtering relies on the correlations between users. It was the first method of collaborative filtering introduced, initially appearing in the GroupLens Usenet article recommender system. In this method, for an active user (user x) who has previous ratings and a user ID available, a group of users with similar past rating behaviors is identified. The ratings from these similar users are then used to predict what might interest the active user [11].

$$s = \{U \times U \rightarrow R\}$$

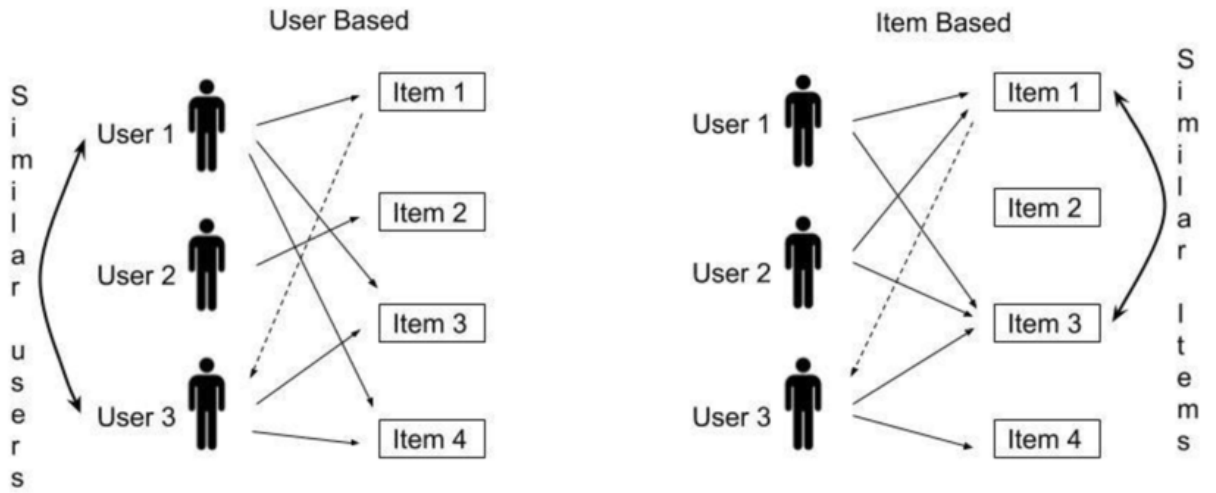


Figure 2.4 – Item Based vs User Based Collaborative Filtering

Item-based collaborative filtering, based on item-item correlations, is one of the most widely developed collaborative filtering techniques. Unlike user-based collaborative filtering, it does not consider user similarity. Instead, it examines the similarity between the rating patterns of items. If two items, a and b, have similar users who like and dislike them, they are considered similar. Consequently, users are anticipated to have similar interests in these items. This approach is similar to content-based filtering in terms of overall structure, but it finds item similarities based on user rating patterns rather than item features [11].

$$s: I \times I \rightarrow R$$

Model-Based Filtering

Model-based approaches construct a predictive model to estimate unknown ratings by learning from observational data. The algorithm creates a model for the user ratings using probabilistic approaches. The parameters of the statistical models are estimated and, unlike the memory-based method, not all available information is included used to make predictions. The collection of ratings is used so that the model

learns user preferences offline. Examples of two popular model-based techniques are Bayesian networks and clustering. Another method is latent variable models and matrix factorization (MF) [2].

On the diagram below there are various approaches and algorithms used in model-based collaborative filtering (CF) along with corresponding Python packages for implementation.

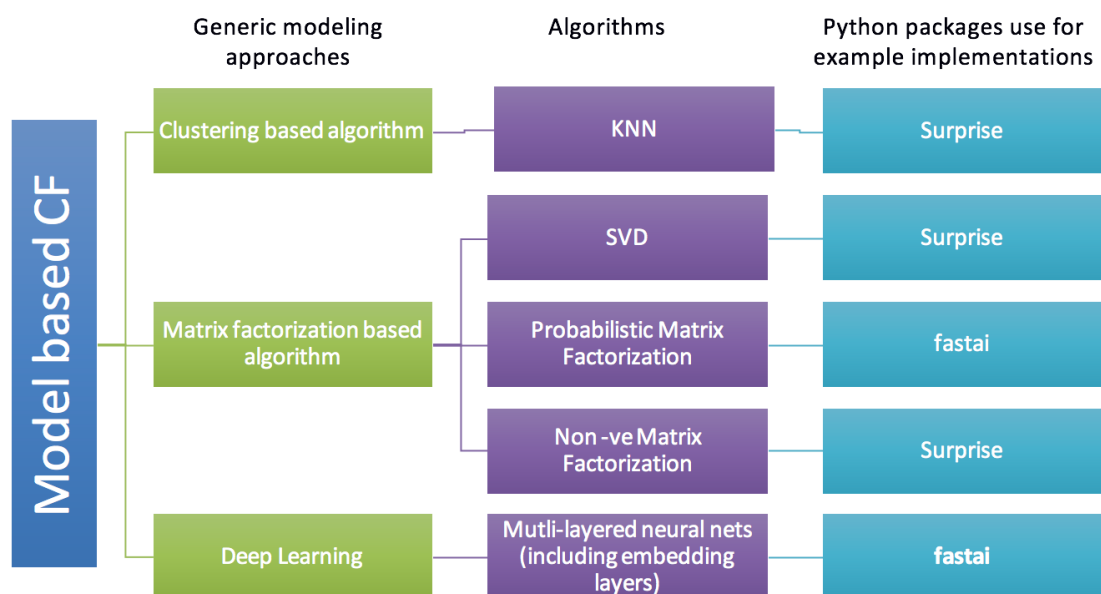


Figure 2.5 – Item Based vs User Based Collaborative Filtering¹

2.1.3 Hybrid Filtering

Hybrid recommender systems combine multiple strategies to achieve better performance than any single approach alone. The most famous example is BellKor's solution, which won the Netflix Prize by combining predictions from 107 different baseline recommender systems. By Burke's survey, the hybrid recommender systems

¹ Image source https://miro.medium.com/v2/resize:fit:1400/1*0vyDJr3urOA6uy-39cr91g.png

are mainly divided into the following classes. They are mixed, switched, weighted, feature-argumentation and meta-level hybrids [12].

Monolithic Hybrid Systems

Monolithic systems, as the name suggests, are composed of a single recommender component that integrates various approaches by pre-processing and combining different knowledge sources. This built-in algorithm modification for handling and pre-processing input data creates a hybrid system. Monolithic systems are further categorized into two types: feature combination and feature augmentation hybrid systems.

In a Feature Combination hybrid system, multiple input data sources are merged into a single recommender algorithm. Conversely, a Feature Augmentation hybrid system takes the output of one algorithm and uses it as the input for another algorithm [2].

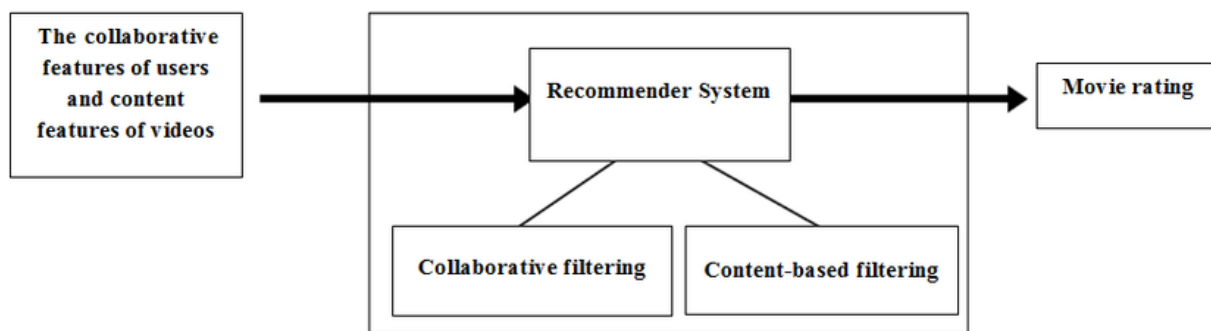


Figure 2.6 – Monolithic Hybrid Systems²

Parallel Hybrid Systems

The paralleled approach requires at least two separate recommender implementations, which are then combined. As shown in Picture 2.7, recommendation lists are created independently. In the hybridization step, these are brought together to create a final output. The parallel approach includes the weighted, switching and mixed methods [11].

Weighted hybrid systems generate recommendations by integrating the weighted sums of the scores from two or more recommendation systems into a single recommendation list. Mixed hybrid systems function similarly, combining the results of various recommender systems and presenting them to the user. Switching hybrid systems alternate between recommendation algorithms based on the quality of the recommendation results and user requirements, ensuring the user receives the best possible recommendations [2].

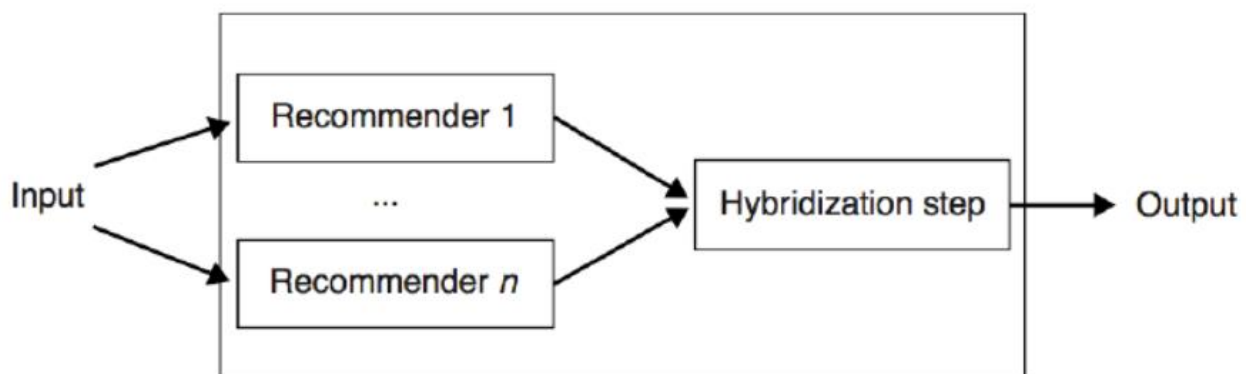


Figure 2.6 – Parallel Hybrid Systems³

Pipelined Hybrid Systems

Pipelined systems generate recommendations through a multi-stage process where various recommendation techniques are sequentially applied. These systems are divided into two types: Cascade systems and Meta-level systems.

Cascade hybrid systems operate on a successor-predecessor principle, where the output of one stage (the predecessor) is refined by the next stage (the successor)

³ Image source <https://upload.wikimedia.org/wikipedia/commons/1/14/Paralelizado.png?20141122193924>

to produce the final recommendations. Meta-level hybrid systems, on the other hand, create a model using one algorithm, and this model then serves as input for another algorithm [13].

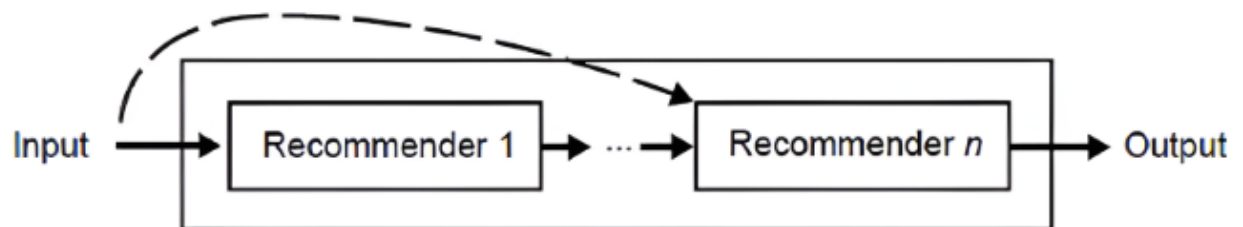


Figure 2.7 – Parallel Hybrid Systems [13].

2.1.4 Knowledge-based Systems

Knowledge-based systems operate on the principle of “Tell me what fits my needs.” Such systems create recommendations that are based on specific domain knowledge. They gather user requirements, compare them with their knowledge base in that domain, and suggest items that are most appropriate and useful for the user, taking into account user preferences [2].

For example, let’s consider online shopping sites. When you want to buy a laptop, the site will ask you to provide your requirements. Based on your input, the system will recommend the most suitable product that not only meets your requirements but is also deemed the best fit for you by the system.

2.1.5 Demographic Systems

Demographic systems, as it clearly says in its name, are based on the user's demographics or the region to which the user belongs. The core concept is that recommendations are connected with the user's demographic information. For example, consider eBay, an online shopping site. Users need to select their region (such as eBay.in, eBay.uk, etc.) for a better experience. Consequently, the system recommends only the products that are available in the selected region, displaying prices in the local currency. This ensures that the only corresponding recommendations would be made for the user's location [13].

2.1.6 Community-based Systems

Community-based systems are based on the principle, “Tell me who your friends are, and I will tell you who you are.” These systems generate recommendations based on the preferences of the user's friends. Unlike to the collaborative filtering, where recommendations are generated within any user (random or friends), community-based systems are operating exclusively on the preferences of the user's social circle [2].

For example, consider Facebook. You are more likely to add someone to your profile if they are suggested by a friend, rather than from the "people you may know" section. This illustrates the essence of community-based systems: we tend to trust recommendations that take our friends' preferences into account over those that consider random individuals.

In this section we've discussed different approaches of Recommendation Systems, in the table below we will find comparison of the pros and cons of above discussed approaches.

Table 2.2 – Comparison of the pros and cons of above discussed approaches
[2]

TYPE	PROS	CONS
Collaborative Filtering	Provides better results when there are many users and ratings available. No knowledge engineering required.	Suffers from cold-start problems for new users and items. Faces sparsity issues.
Content-Based Filtering	Does not rely on other users' preferences. Allows comparison between items.	Difficult to make valid recommendations for users with many purchases or varying tastes. Cold start for new users. Lacks surprise elements.
Knowledge-Based Systems	Offers deterministic recommendations with assured quality. No cold start.	High cost of knowledge acquisition from domain experts and users. Generally static.
Hybrid Systems	Avoids some shortcomings of other methods.	Most datasets do not allow comparison of different recommendation paradigms.
Demographic Systems	Recommendations are valid only for a particular region, avoiding invalid suggestions.	Relatively less research in this area.
Community-Based Systems	Enables simple and comprehensive data acquisition related to users' social relations.	Recommendations are not always accurate.

2.2 Challenges in Recommendation systems

2.2.1 Cold-start Problem

The cold start problem occurs when it is quite challenging to generate accurate recommendations for a new user or item due to a lack of historical data. For example, a father learned about his teenage daughter's pregnancy through targeted ads. The company had accurately predicted her due date based on the products she purchased, despite having no prior information about her pregnancy. Example of cold start is below.

Table 2.3 – Cold Start [13]

	Item1	Item2	Item3	Item4	Item5
New User	?	?	?	?	?
User1	3	3	1	4	2
User2	4	2	4	4	5
User3	4	1	2	5	4
User4	5	3	4	3	2

2.2.2 Sparsity

Sparsity comes from the word "sparse," which has a meaning "scattered." Considering the context of recommender systems, sparsity refers to the irregular, insufficient, or highly variable user ratings. It is one of the main limitations that are faced by these systems. The main reason of sparsity is that most users do not provide ratings, and those that are available are often too scattered or sparse. For instance, when rating a mobile app, many users are ignoring the request to rate, and those who are really rating might either give a very positive or very negative rating. As a result, the available ratings are quite scattered. This is a significant problem for collaborative filtering, which relies on user ratings to make recommendations.

Table 2.4 – Sparsity [13]

	Item1	Item2	Item3	Item4	Item5
Sabrina	?	4	2	5	?
User1	?	1	?	?	?
User2	?	?	?	?	?
User3	?	?	2	4	?
User4	3	?	?	3	?

2.2.3 Scalability

Scalability means the system's ability to handle an increasing amount of data effectively. While recommender systems perform well with small datasets, managing real-world datasets that grow extensively could be a Challenge. Although there are algorithms that are designed to handle massive and dynamic datasets, their results are not always accurate.



Figure 2.8 – Issue with Scalability [13].

2.2.4 Shilling attacks

Recommender systems can be targeted by two primary types of attacks: push attacks and nuke attacks. Push attacks aim to increase product ratings by creating fake profiles that give false positive reviews, thereby boosting the product's popularity. On the other hand, nuke attacks seek to decrease the popularity of competitors' products by providing false negative reviews. These attacks can be categorized based on the attacker's rating strategy into several types, including probe

attacks, random attacks, average attacks, popular attacks, segment attacks, love/hate attacks, sampling attacks, and perfect knowledge attacks.

2.2.5 GraySheep Problem

The gray sheep problem means users who demonstrate inconsistent behavior, making it difficult to predict their preferences accurately. Such users have undefined preferences and may like one type of content at one moment and the exact opposite at another. For instance, if a user likes both romantic and action movies, a recommender system might generate a recommendation like romantic movies to another user with similar action movie preferences. However, these romantic movie recommendations might not interest the second user, what leads to irrelevant suggestions. As a result, this inconsistency decreases the efficiency of recommender systems.

2.3 Data Collection

To build recommender systems—or machine learning systems in general—data is required to base the model development on. This data can come from various sources and be of different types. For content-based filtering (CBF), information about the entities to be recommended is needed to develop a recommender model based on content attributes. However, user data is crucial, especially for collaborative filtering (CF). Different types of user data can be distinguished (see Table 5). In data collection for user data, known as preference data, there are two main sources: explicit and implicit data [14].

To test the learned models, the available data is divided into a training dataset and a test dataset. The models are built and trained using the training data, and then evaluated and validated with the test data.

Table 2.5 – Examples of Data collection

Data Type	Description
Rating Data	Rating Scores (discrete or continuous rating data); Latent Comments (e.g., excellent, good, bad, worse); Likes and Dislikes;
Behaviour Pattern Data	Duration of Browsing, Number of Clicks; Links to Websites; Saving, Printing, Scrolling, Deleting, Closing, Refreshing Websites; Selection, Editing, Search, Copying, Inserting, Marking, and Downloading of Web Content;
Transaction Data	Purchase Data, Number of Purchases; Prices; Discounts;

2.3.1 Explicit Data

Explicit data refers to feedback provided directly by the user. This user input can take various forms: likes and dislikes, a rating system with a fixed scale (e.g., a rating scale from 1 to 5), forms to fill out, or text reviews (e.g., user reviews on streaming platforms).

The advantage of explicit data is that the feedback comes directly from the user. However, the drawback is that such information is often not available in sufficient quantities. Users rarely take the time to rate something or provide personal information. Additionally, there may be discrepancies between what the user reports and what is actually true [14].

In such preference data, the goal of a recommender system is to predict the ratings for items that a user has not yet rated.

2.3.2 Implicit Data

Implicit data refers to feedback that does not require additional user input. Examples include transaction data (e.g., what the user watched and when) and other activity or click data: which content they viewed and how often, what they saved, what they searched for, and what they added to their watchlist. This type of data reflects the user's activity.

The advantage of implicit data collection is that much more data is available. However, according to Ekstrand et al., implicit data is considered to be noisier or less accurate. Nevertheless, it can provide a more accurate picture of a user's interests over the long term. Systems that collect implicit feedback are more commonly used in practice. In streaming platforms, viewing history and watch time are the main sources of preference data. A recommender system is not limited to using only explicit or implicit data but can use both types as sources of information.

2.4 Recommendation Algorithm [2⁴]

2.4.1 Computing User Similarity

Pearson Co-relation

This algorithm is used to find the similarity between two users based on their ratings. The values lie between -1 (strong negative co-relation) to +1 (strong positive co-relation) [2]. It is given by:

$$\text{sim}(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}},$$

⁴ Here and further

where a, b – users

$r_{a,p}$ – rating of user a for product p ;

$\bar{r}_a, \bar{r}_b$ – average user rating;

P – set of products rated by both a and b .

Constrained Pearson Co-relation

This algorithm is used when the co-relation coefficient between two users increases as a result of either positive or negative ratings from the user. If the rating score is below the midpoint of the scaling scheme, it is considered as negative rating while a score above the midpoint is considered as a positive rating [2]. It is given by:

$$\text{sim}(a, b) = \frac{\sum (p_{ai} - mp)(p_{bi} - mp)}{\sqrt{\sum (p_{ai} - mp)^2} \sqrt{\sum (p_{bi} - mp)^2}},$$

where mp – midpoint of rating scale.

Spearman Co-relation

It is a non-parametric method that computes a measure of co-relation between ranks. It is similar to that of Pearson co-relation except that it uses ranks instead of ratings [2]. It is given by:

$$\text{sim}(a, b) = \frac{\sum (\text{rank}_{ai} - \overline{\text{rank}_a})(\text{rank}_{bi} - \overline{\text{rank}_b})}{\sqrt{\sum (\text{rank}_{ai} - \overline{\text{rank}_a})^2} \sqrt{\sum (\text{rank}_{bi} - \overline{\text{rank}_b})^2}}$$

Cosine Similarity

It is a vector based approach based on linear algebra. Users are considered as vectors and similarity between them is measured by the cosine distance or dot

product of the angles between the rating vectors [2]. It is given by:

$$\text{sim}(a, b) = \frac{r_a \cdot r_b}{|r_a|_2 |r_b|_2}$$

2.4.2 Computing Item Similarity

Cosine Similarity

It is the most popularly used metric for calculating similarity between items. It is the dot product or cosine product of the angles between the rating vectors, where the vectors are the items itself [2]. It is given by:

$$\text{sim}(i, j) = \frac{r_i \cdot r_j}{|r_i|_2 |r_j|_2}$$

Conditional Probability

It is generally used in online shopping sites where the purchase histories of the customer are available [2]. It is given as:

$$\text{sim}(i, j) = \Pr_{r_B}[j \in B \mid i \in B],$$

where B – User's purchase history.

2.4.3 For Neighbourhood Generation

K Nearest Neighbour Approach

K nearest neighbour approach is a simple algorithm used as a non-parametric technique to calculate the similarity measure or distance between the neighbours (users, items or products). The neighbour with the smallest distance measure will be the k-nearest neighbour and the recommendations would be made based on that [2]. Nearest neighbour can be found out by calculating the distance between all the neighbours by using:

- euclidean distance

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2};$$

- manhattan distance

$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}.$$

Association Rule Mining

It is a common data-mining technique used to identify rule-like relationship patterns, co-relations and associations from the datasets. It is generally a combination of if/then statements that helps in finding relationships between varying products or users. Simply put, it finds the rules which support the relationship of a particular item with others, to make future predictions. For example, a rule can be “If a user buys bread, then he or she will buy butter or jam also in most of the cases”. The measures of association rule are: support and confidence [2].

2.5 Metrics for Evaluating Recommender Systems [2⁵]

In this section, will be discussed different parameters on the basis of which, can be measured the performance of a recommender system.

2.5.1 Coverage

Coverage for a recommender system must always be high, i.e. higher value of coverage indicates that the maximum portion of items is being recommended³². It can be item space coverage or user space coverage [2].

$$\text{Coverage} = \frac{N_d}{N},$$

where N_d – no. of distinct objects;

N – Total no. of objects.

2.5.2 Precision

Precision is a measure of goodness that determines the proportion of recommended items from the available ones [2]. It is given by:

$$\text{Precision} = \frac{t_p}{t_p + f_p},$$

where T_p – true positive;

⁵ Here and further

F_p – false positive.

2.5.3 Recall

Recall is a measure of completeness that gives the proportion of relevant recommendations made out of all the relevant ones [2].

$$\text{Recall} = \frac{t_p}{t_p + f_n},$$

where f_n – False negative.

2.5.4 Hit Rate

Hit Rate at K calculates the share of users for which at least one relevant item is present in the K [17].

$$\text{HR} = \frac{|U_{\text{hit}}^L|}{|U_{\text{all}}|},$$

where $|U_{\text{hit}}^L|$ – number of users for which the correct answer is included in top(L) recommendation list;

$|U_{\text{all}}|$ – total number of users in the test dataset.

2.5.5 Leave-One-Out Cross-Validation

Leave-one-out cross-validation (LOOCV) is particularly useful for evaluating the performance of machine learning models when the dataset or category value is small. In LOOCV, one data sample from the n data samples is taken as the test set, and the remaining $n-1$ data samples are used as the training set. This process is repeated for each data sample, resulting in n training models.

LOOCV provides relatively stable results by testing all samples, in contrast to the validation set approach used in other cross-validation methods such as 10-fold or k -fold. Because it uses every sample as a test set once, LOOCV is a nearly unbiased and reliable method for estimating model performance, provided that the training and testing sets are drawn from the same distribution [18].

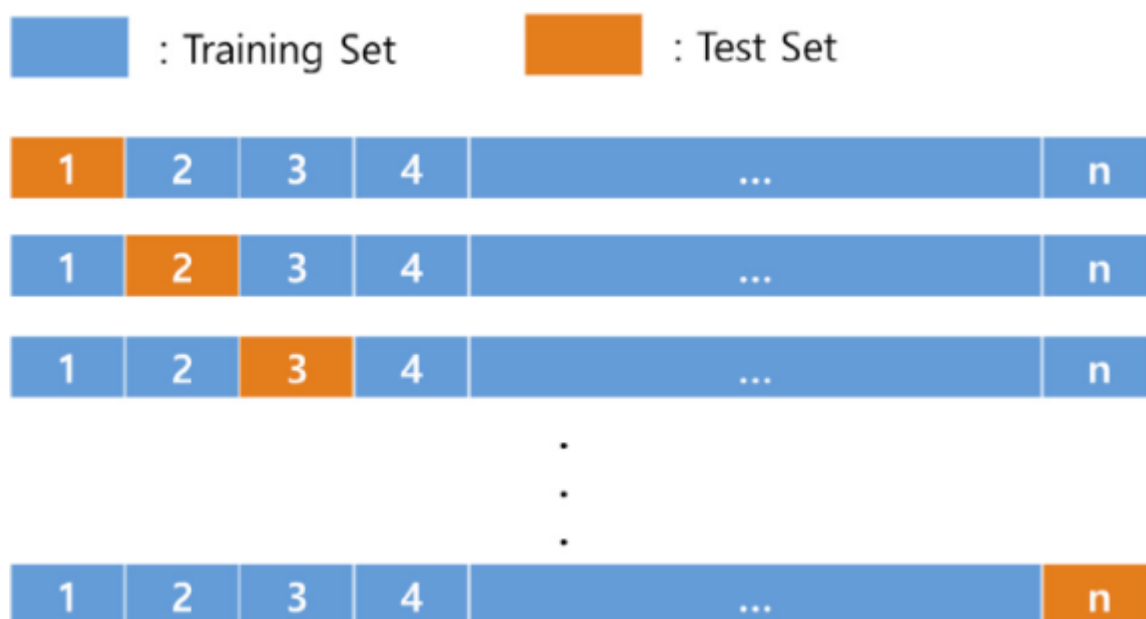


Figure 2.9 – Issue with Scalability [18].

2.6 Learning Types

2.6.1 Unsupervised

Unsupervised learning employs machine learning algorithms to analyze and cluster unlabeled datasets, identifying hidden patterns without human intervention, hence the term “unsupervised”.

These models are typically used for three main tasks [16]:

- clustering: this data mining technique groups unlabeled data based on similarities or differences. For instance, K-means clustering assigns similar data points into groups, where the value of K indicates the number and granularity of the groups. Clustering is useful in applications such as market segmentation and image compression;
- association: this method finds relationships between variables in a dataset using various rules. Association techniques are often used in market basket analysis and recommendation engines, such as those suggesting additional items based on previous purchases;
- dimensionality reduction: this technique is applied when a dataset has too many features (or dimensions). It reduces the number of inputs to a manageable size while maintaining data integrity. Dimensionality reduction is often used in the preprocessing stage, such as when autoencoders remove noise from visual data to enhance image quality.

2.6.2 Supervised

Supervised learning is a machine learning approach characterized by its use of labeled datasets. These datasets are designed to train or "supervise" algorithms, enabling them to classify data or predict outcomes accurately. By using labeled inputs and outputs, the model can measure its accuracy and improve over time.

Supervised learning can be divided into two main types of problems in data mining: classification and regression [16]:

- classification: this type of problem involves using an algorithm to accurately assign test data to specific categories, such as distinguishing between apples and oranges. In practical applications, supervised learning algorithms can classify spam emails separately from your inbox. Common classification algorithms include linear classifiers, support vector machines, decision trees, and random forests;
- regression: this type of supervised learning method uses an algorithm to understand the relationship between dependent and independent variables. Regression models are useful for predicting numerical values based on various data points, such as sales revenue projections for a business. Popular regression algorithms include linear regression, logistic regression, and polynomial regression.

Conclusions to section 2

In this chapter, we explored the key methodologies in recommender systems, including content-based filtering, collaborative filtering, and hybrid filtering. We also discussed additional approaches such as knowledge-based, demographic-based, and community-based systems.

Data collection techniques, both explicit and implicit feedback, were examined, along with different learning approaches and key evaluation metrics. Challenges such as the cold-start problem, sparsity, scalability, shilling attacks, and the gray sheep problem were also highlighted.

This chapter provides a foundational understanding of the strategies and challenges involved in developing effective recommender systems.

3 PRODUCT DEVELOPMENT AND EVALUATION PROCESS

The primary objective of this project is to develop a comprehensive movie recommendation system utilizing a combination of modern technologies and libraries. This system aims to provide personalized movie recommendations to users based on their past interactions and preferences. The development process involves several key components: the frontend, backend, recommendation engine, and external API integration. Each component plays a vital role in ensuring the system's functionality, accuracy, and user satisfaction.

3.1 Development Process Overview

The development process involves several key steps, each critical to the successful implementation of the recommendation system:

- initialization and setup: firstly, set up the development environment for the frontend, backend, and recommendation engine, then configure the necessary libraries and frameworks, including React, Flask, Apache Spark, and the Surprise library;
- data loading and preparation: first load user-item interaction data from a CSV file, then transform the data into a suitable format using Spark's DataFrame API and prepare it for model training;
- model training and evaluation: first train the ALS model using the prepared data, then optimizing parameters such as the number of iterations and regularization factor, evaluate the model's performance using RMSE and other relevant metrics to ensure its accuracy;
- building the backend API: develop Flask endpoints to handle user requests and interface with the recommendation engine, then implement

CORS (Cross-Origin Resource Sharing) to enable communication between the frontend and backend;

- frontend development: firstly, create React components to handle user inputs and display movie recommendations, then integrate the frontend with the backend API to fetch and display recommendations dynamically;
- integration with TMDb API: first fetch additional movie details and images from the TMDb API to enrich the recommendations, then display the fetched images and metadata in the frontend for an enhanced user experience;
- testing and deployment: test the system end-to-end to ensure all components work seamlessly together, then deploy the application on a suitable platform, ensuring it is accessible to users.

3.2 Choosing programming languages, frameworks and libraries for implementation

3.2.1 Frontend

JavaScript is essential for creating dynamic and interactive web applications and is supported by all modern web browsers, ensuring compatibility across devices. It has a vast ecosystem of libraries and frameworks like React, Angular, and Vue.js that simplify complex tasks and enhance productivity. JavaScript excels in event-driven programming, allowing for responsive user interfaces, and supports asynchronous operations, crucial for tasks like fetching data from APIs. Additionally, its large and active community provides extensive resources and support [19].

React, developed by Facebook, is a popular JavaScript library for building user interfaces. It uses a component-based architecture, promoting code reusability and maintainability. React's Virtual DOM optimizes rendering performance by updating

only the parts of the DOM that change, leading to faster rendering. Its declarative syntax simplifies UI development by allowing developers to describe the UI state, and React manages the updates. React has a rich ecosystem of tools and libraries, enhancing productivity, and is widely adopted in the industry. It also supports server-side rendering, improving initial load times and SEO [20].

3.2.2 Backend

In this project, Python and Flask were chosen as the primary technologies for backend development due to their inherent advantages. Python's simplicity and readability facilitate the writing of clean and maintainable code, while its versatile standard library and extensive ecosystem of third-party packages provide the tools necessary to build robust backend systems [21].

Flask, being a lightweight and flexible web framework, complements Python's strengths by offering a modular architecture that allows developers to build web applications with minimal overhead. Flask's minimalist approach ensures that only essential components are included, which can be extended with additional functionality as required. This combination of Python and Flask creates a powerful and efficient backend that can handle various tasks, from handling HTTP requests to interfacing with databases and other services [22].

3.2.3 Recommender engine

Python is an excellent choice for building a recommender engine due to its simplicity, readability, and extensive libraries. It is known for its clear and concise

syntax, which makes it easier to write and maintain code, especially for complex algorithms involved in recommendation systems. Python supports various programming paradigms, including procedural, object-oriented, and functional programming, providing flexibility in designing the recommender system [21]. Python has a vast ecosystem of libraries and frameworks that facilitate data manipulation, machine learning, and scientific computing. Libraries such as NumPy, pandas, and SciPy provide powerful tools for data processing and analysis, which are essential in preparing and preprocessing the data for the recommendation engine. Moreover, Python's extensive community and comprehensive documentation make it easier for developers to find resources and support, enhancing the development process.

The Surprise library is specifically designed for building and analyzing recommender systems. It provides various ready-to-use algorithms for collaborative filtering, including popular methods like Singular Value Decomposition (SVD) and k-Nearest Neighbors (k-NN). The library simplifies the process of training and evaluating different algorithms, allowing developers to focus on improving the model's performance [23]. Surprise supports loading data from various sources, including built-in datasets and custom data files, making it highly versatile for different data formats. It also offers tools for cross-validation and performance evaluation, which are crucial for assessing the effectiveness of the recommender engine. The library's ease of use and comprehensive functionalities make it a preferred choice for developing recommendation systems.

Apache Spark is a powerful distributed computing system that is well-suited for handling large-scale data processing tasks. It is particularly beneficial for building a recommender engine for movies due to its ability to efficiently process and analyze large datasets. Spark's in-memory computing capabilities significantly enhance the speed of data processing, making it ideal for real-time recommendations [24]. Spark includes MLlib, a scalable machine learning library that provides various algorithms for collaborative filtering, including the Alternating Least Squares (ALS) algorithm. ALS is highly effective for building recommender systems and is optimized for large

datasets, making it suitable for movie recommendation tasks where data volume can be substantial [25].

3.2.4 Environment

Visual Studio Code (VS Code) is an excellent choice for this project due to its powerful features, extensive customization options, and strong support for the technologies we are using. It provides intelligent code completion, syntax highlighting, and easy navigation through the codebase, which enhances productivity and reduces development time. The integrated terminal allows developers to run commands, scripts, and interact with the backend, frontend, and Spark environments without leaving the editor [26]. VS Code has a rich ecosystem of extensions that support Python, JavaScript, Flask, React, and Spark. These extensions offer features like linting, debugging, and code formatting, streamlining the development process. The built-in debugging tools for both Python and JavaScript help in quickly identifying and fixing issues, making the development and testing phases more efficient.

3.2 Data collecting

3.2.1 Datasets

In the development of our recommender system for a streaming platform, we utilized multiple datasets to capture user interactions and movie metadata.

The dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. This data were created by 610

users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018 [15].

The dataset (ml-25m) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 25000095 ratings and 1093360 tag applications across 62423 movies. These data were created by 162541 users between January 09, 1995 and November 21, 2019. This dataset was generated on November 21, 2019 [15].

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

These datasets provide a comprehensive foundation for building and evaluating various recommendation algorithms. The datasets include user ratings, movie metadata, user-generated tags, and links to external databases. Each dataset is described in detail below:

1. Movie Metadata (movies.csv).

The movies.csv file contains detailed information about each movie available in the dataset. It includes the following columns:

- movieId: a unique identifier for each movie;
- title: the title of the movie, often including the release year;
- genres: the genres associated with the movie, separated by a | character.

This dataset is crucial for content-based filtering, allowing the system to recommend movies based on their attributes.

Table 3.1 – Example of movie.csv

movieId	title	Genres
1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
2	Jumanji (1995)	,Adventure Children Fantasy

Usage:

- content-based filtering: utilizes genre information to recommend movies similar to those a user has liked in the past;
- metadata enrichment: enhances movie profiles by linking them with external data sources through links.csv.

2. User Ratings (ratings.csv).

The ratings.csv file captures user ratings for movies, which are pivotal for collaborative filtering. It includes the following columns:

- **userId**: a unique identifier for each user;
- **movieId**: the ID of the movie rated by the user;
- **rating**: the rating given by the user, typically on a scale from 0.5 to 5.0. This explicit feedback reflects the user's preference for the movie;
- **timestamp**: the timestamp when the rating was provided, which can be used to analyze rating trends over time.

This dataset is essential for understanding user preferences and computing similarities between users and items.

Table 3.2 – Example of ratings.csv

userId	movieId	rating	timestamp
1	1	4.0	964982703
2	3	4.5	964981247

Usage:

- collaborative filtering: leverages user ratings to identify similar users and movies, enabling personalized recommendations;
- temporal analysis: uses timestamps to understand changes in user preferences over time and to model trends.

3. User-Generated Tags (tags.csv).

The tags.csv file contains tags provided by users for movies, adding another layer of descriptive metadata. It includes the following columns:

- **userId**: a unique identifier for each user;
- **movieId**: the ID of the movie tagged by the user;
- **tag**: the tag provided by the user, which is typically a keyword or phrase describing an aspect of the movie;
- **timestamp**: the timestamp when the tag was given.

Tags help in enhancing the content-based filtering by incorporating user-generated descriptors and insights into the movies.

Table 3.3 – Example of ratings.csv

userId	movieId	tags	timestamp
1	1	Anime, cool	964982703
2	3	Fiction, funny	964981247

Usage:

- **enhanced metadata**: uses tags to enrich movie descriptions and improve the granularity of content-based filtering;
- **user preferences**: captures unique user perspectives and preferences that may not be evident from ratings alone.

3.2.2 Data Preparation for Collaborative Filtering with Apache Spark

The data preparation process is a critical step in developing a collaborative filtering recommendation system using Apache Spark. This process involves several stages, including initializing the Spark environment, loading and transforming the data, and finally, splitting the data into training and test sets. Each of these steps is essential to ensure the data is in the correct format for model training and evaluation.

1. **Initializing Spark Session**. The first step involves setting up the Spark session, which serves as the entry point for using Spark's functionalities.

This initialization is crucial as it configures the Spark context for subsequent data processing and machine learning tasks.

2. **Loading Data.** The dataset used in this study is loaded from a CSV file containing user-item interaction data. The data loading process employs Spark's DataFrame API, which provides a high-level abstraction for reading structured data. The CSV file is read into a Resilient Distributed Dataset (RDD) with the option `header=true`, indicating that the first row of the file contains the column names.
3. **Data Transformation.** Once the data is loaded into an RDD, it undergoes a transformation to convert each row into a structured format. This transformation utilizes Row objects, which allow the assignment of specific data types to each column, such as integers for user and item identifiers and floats for ratings. This step is essential for ensuring that the data conforms to the expected schema required by Spark's machine learning algorithms.
4. **Creating DataFrame.** The transformed RDD is then converted into a DataFrame. DataFrames in Spark offer a more efficient and convenient way to manipulate structured data compared to RDDs. They provide a range of high-level operations and optimizations that facilitate data preparation and processing.
5. **Splitting Data.** To evaluate the performance of the recommendation model, the data is split into training and test sets. This study uses an 80/20 split, where 80% of the data is allocated to the training set and the remaining 20% to the test set. The `randomSplit` method is employed for this purpose, ensuring that the data is randomly divided, which helps in achieving a representative and unbiased split.

3.3 Explanation of the choice of algorithm and its implementation

Collaborative Filtering and Matrix Factorization using ALS

A sparse matrix R can be constructed based on the data of user-item relationships and their respective ratings. The scores can be obtained from explicit user ratings or inferred from user behaviours. Collaborative Filtering methods utilize this matrix R to generate recommendations.

Initial Sparse Matrix:

movieId	1	2	3	4	5	6	7	...	193573	193579	193581	193583	193585	193587	193609
userId	1	2	3	4	5	6	7	...	193573	193579	193581	193583	193585	193587	193609
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	2.5	0.0	0.0	0.0	0.0	0.0	2.5	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
607	4.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
608	2.5	2.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
609	3.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
610	5.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 3.1 – Initial sparse matrix in our application.

Matrix Factorization

The goal of matrix factorization is to decompose the utility matrix R into two lower-dimensional matrices: the user latent matrix U and the item latent matrix P . This decomposition aims to approximate the original matrix R as closely as possible.

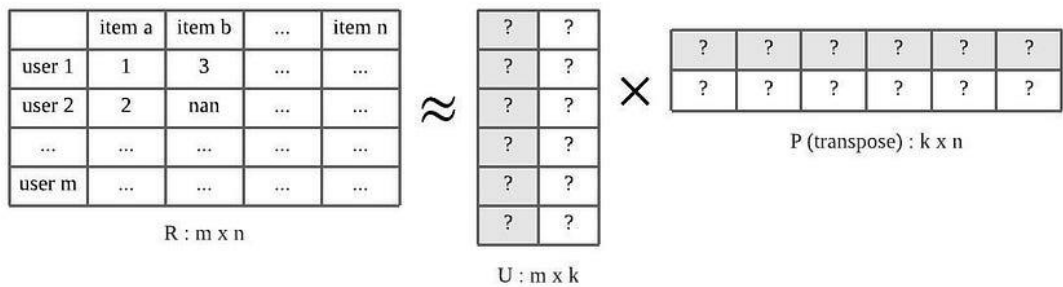


Figure 3.2 – Matrix Factorization [27].

There are several methods to factorize the utility matrix, such as Singular Value Decomposition (SVD) and Probabilistic Latent Semantic Analysis (PLSA). Among these, the Alternating Least Squares (ALS) algorithm is widely used for collaborative filtering due to its efficiency and effectiveness.

Alternating Least Squares (ALS)

The ALS model is one of the most popular methods in collaborative filtering. To understand the mathematics behind the model, we define the objective function using the Root Mean Square Error (RMSE) as the loss function:

$$\text{RMSE} = \sqrt{\frac{(\text{real} - \text{prediction})^2}{n}}$$

Assume there are $\mathbf{m}$ users and $\mathbf{n}$ items, $\mathbf{R} = \mathbf{m} * \mathbf{n}$, $\mathbf{U} = \mathbf{m} * \mathbf{k}$, $\mathbf{P} = \mathbf{n} * \mathbf{k}$, where $\mathbf{k}$ is the latent factors [27].

$$\begin{aligned} \text{loss} &= \min(\text{real} - \text{prediction})^2 \\ &= \min(\mathbf{R} - \mathbf{U} \cdot \mathbf{P}^T)^2 \\ &= \min \sum_{x,y} (\mathbf{R}_{x,y} - \mathbf{U}_x \cdot \mathbf{P}_y^T)^2 \end{aligned}$$

In order to avoid overfitting, we add l2 norm to our objective function, such that

$$\text{loss} = \min \sum_{x,y} (\mathbf{R}_{x,y} - \mathbf{U}_x \cdot \mathbf{P}_y^T)^2 + \lambda(|\mathbf{U}|^2 + |\mathbf{P}|^2)$$

Next, we take the partial differentiation respect to $\mathbf{U}$ and $\mathbf{P}$.

$$\begin{aligned}
& \frac{\partial \text{loss}}{\partial U} = 0 \\
& = \frac{\partial}{\partial U} \sum_{x,y} (R_{x,y} - U_x \cdot P_y^T)^2 + \lambda(|U|^2 + |P|^2) = 0 \\
& = -2 \sum_{x,y} (R_{x,y} - U_x \cdot P_y^T) P_y + 2\lambda U_x = 0 \\
& = -(R_x - U_x^T \cdot P^T) P + \lambda U_x^T = 0 \\
& = U_x^T = R_x P (P^T P + \lambda I)^{-1}
\end{aligned}$$

By applying similar process, we can find the loss differentiation respect to P.

$$\begin{aligned}
& \frac{\partial \text{loss}}{\partial P} = 0 \\
& = \frac{\partial}{\partial P} \sum_{x,y} (R_{x,y} - U_x \cdot P_y^T)^2 + \lambda(|U|^2 + |P|^2) = 0 \\
& = P_y^T = R_y U (U^T U + \lambda I)^{-1}
\end{aligned}$$

By applying these derivatives, we derive the update equations for U and P. By fixing one matrix, we can optimize the other. This iterative process of alternately optimizing U and P continues until convergence, thus optimizing the factorization of the utility matrix.

After the ALS algorithm completes its iterative optimization process, the result is two matrices: the user matrix U and the item matrix P. These matrices represent the latent factors for users and items, respectively. The dot product of these matrices yields a predicted user-item interaction matrix R.

Matrix with Predictions:													
movieId	1	2	3	4	5	6	...	193567	193571	193579	193583	193585	193609
userId													
1	4.716318	3.947861	3.980740	2.541424	2.825013	3.956821	...	2.264647	3.019529	2.642088	2.642088	2.642088	1.990592
2	3.266056	3.455148	2.857411	1.184005	2.547020	2.541693	...	2.136432	2.848577	2.492505	2.492505	2.492505	-0.331769
3	1.044265	0.949242	-0.267962	-0.087593	-0.622673	-0.453146	...	1.320345	1.760459	1.540402	1.540402	1.540402	0.322596
4	3.019403	2.474138	2.985378	1.948838	1.111179	2.634753	...	0.716400	0.955200	0.835800	0.835800	0.835800	1.403863
5	3.925127	3.277599	3.854559	2.859402	2.206781	3.874365	...	0.808377	1.077836	0.943106	0.943106	0.943106	0.414420
...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	3.952981	2.949383	3.459051	2.517517	2.380092	3.894856	...	1.595660	2.127546	1.861603	1.861603	1.861603	2.148504
607	4.168560	3.775840	3.617038	2.858983	2.869177	4.881341	...	1.860770	2.481026	2.170898	2.170898	2.170898	1.686550
608	3.538592	2.875424	3.395823	1.341772	2.474756	3.435305	...	1.902053	2.536070	2.219062	2.219062	2.219062	0.685098
609	3.348610	2.983785	2.829046	1.828508	2.646245	3.860767	...	1.488974	1.985300	1.737137	1.737137	1.737137	0.597628
610	4.122265	3.659878	2.971406	2.599041	2.570124	4.460546	...	1.876818	2.502424	2.189621	2.189621	2.189621	2.207698

Figure 3.3 – Matrix with predictions in our application.

The figure 3.4 shows a comparison between real and predicted ratings for a recommendation system. Each row represents a user-movie interaction, with columns for user ID, movie ID, actual rating, timestamp, and predicted rating. The actual ratings range from 3.0 to 4.5, while the predicted ratings range from 3.12 to 4.02. The predictions are generally close to the actual ratings, indicating the model's effectiveness. For instance, User 148's predictions are within 0.12 to 0.78 points of the real ratings. This suggests that the ALS model used is accurately capturing user preferences.

userId	movieId	rating	timestamp	prediction
148	1197	3.0	1482548478	3.7206614
148	5618	3.0	1482548682	3.6516957
148	7153	3.0	1482548769	3.6586905
148	31658	4.0	1482548683	3.5877557
148	40629	5.0	1482548578	3.123804
148	79702	4.0	1482548751	3.3001912
148	89745	4.0	1482548710	3.4372387
148	108932	4.0	1482548720	3.2640715
148	115617	3.5	1482548495	3.5196888
148	122886	3.5	1482548686	3.3863494
148	134853	4.0	1482548516	3.4199908
463	527	4.0	1145460304	4.018101
463	2019	4.0	1145460514	3.9894311
463	6947	3.5	1145460389	3.5718727
463	7320	4.0	1145460102	3.313225
471	527	4.5	1496671869	3.755083
471	4973	4.0	1496669566	3.8026104
471	8961	3.5	1496671872	3.4460917
471	78499	4.5	1496671862	3.6654375
471	79132	4.0	1496668936	3.5582807

Figure 3.4 – Table with real and predicted ratings

3.4 Evaluation of ALS Algorithm for Recommendation System

This chapter evaluates the performance of the ALS algorithm using various metrics, explores how these metrics change with different ALS parameter settings, and provides us conclusions based on the observed results.

3.4.1 Evaluation Metrics

To assess the performance of the ALS algorithm, we use the following metrics.

1. Root Mean Square Error (RMSE): measures the square root of the average squared differences between predicted and actual ratings. Lower RMSE values indicate better predictive accuracy.
2. Mean Absolute Error (MAE): measures the average magnitude of errors in predictions without considering their direction. Lower MAE values indicate better performance.
3. R-squared (R^2): represents the proportion of variance in the dependent variable that is predictable from the independent variables. Higher R^2 values, closer to 1, indicate a better fit.

3.4.2 Methodology

We evaluate the ALS algorithm's performance using a dataset of movie ratings. The dataset is divided into training and test sets, and the ALS model is trained on the training set and evaluated on the test set. We vary the key parameters

of the ALS algorithm, namely maxIter, regParam, and rank, and observe their impact on the evaluation metrics.

3.4.3 Experimental Setup

The experiments are conducted with the following ALS parameter settings:

- maxIter: Number of iterations (5, 10, 20);
- regParam: Regularization parameter (0.01, 0.1, 0.5);
- rank: Number of latent factors (10, 20, 50).

We use cross-validation to ensure qualitative evaluation and to find the optimal parameter settings.

3.4.4 Results and Analysis

Impact of maxIter

Increasing the number of iterations generally leads to better convergence, as shown in the metrics:

Table 3.4 – Impact of maxIter

maxIter	RMSE	MAE	R2
5	0.907	0.726	0.765
10	0.865	0.692	0.773
20	0.840	0.675	0.784

Impact of regParam

Regularization helps prevent overfitting. Optimal regularization balances bias and variance:

Table 3.5 – Impact of regParam

regParam	RMSE	MAE	R2
0.01	0.907	0.726	0.782
0.1	0.875	0.705	0.794
0.5	0.950	0.780	0.723

Impact of rank

The number of latent factors determines the model's complexity. More factors can capture more interactions but may overfit:

Table 3.6 – Impact of rank

Rank	RMSE	MAE	R2
10	0.907	0.726	0.781
20	0.870	0.700	0.796
50	0.860	0.690	0.812

The evaluation of the ALS algorithm indicates that increasing the number of iterations (maxIter) generally improves the model's performance metrics. However, excessively high values may lead to overfitting. The regularization parameter (regParam) needs careful tuning to balance bias and variance. A lower regularization parameter often fits the training data well but risks overfitting. The number of latent factors (rank) also plays a crucial role, where higher values capture more complex interactions but may require more data to avoid overfitting.

In conclusion, optimal performance of the ALS algorithm is achieved through careful tuning of maxIter, regParam, and rank parameters. Cross-validation helps identify the best combination of parameters, ensuring robust and accurate recommendations.

By systematically evaluating the ALS algorithm with various parameters, we gain insights into its strengths and limitations, enabling us to build more effective recommendation systems.

3.5 Integration of User Interface with Backend Recommendation Engine

In this chapter, we describe the development and integration of a user interface (UI) with a backend recommendation engine, forming a cohesive system that delivers personalized movie recommendations to users. The UI, built with React, communicates with a backend powered by Flask and Apache Spark. The backend processes user data, generates recommendations using the Alternating Least Squares (ALS) algorithm, and fetches movie posters from The Movie Database (TMDB) API.

3.5.1 Frontend Development

The frontend, developed using React, consists of three main components: App, Carousel, and CustomInput. The App component manages the state and fetches recommendations from the backend. The CustomInput component allows users to input their ID to get recommendations.



Figure 3.5 – Input to generate recommendations for a specific user

The Carousel component displays the recommended movies and fetches movie posters from TMDB. State management and API integration are handled using React hooks. Let's look at the example of recommendations of User with id #10.

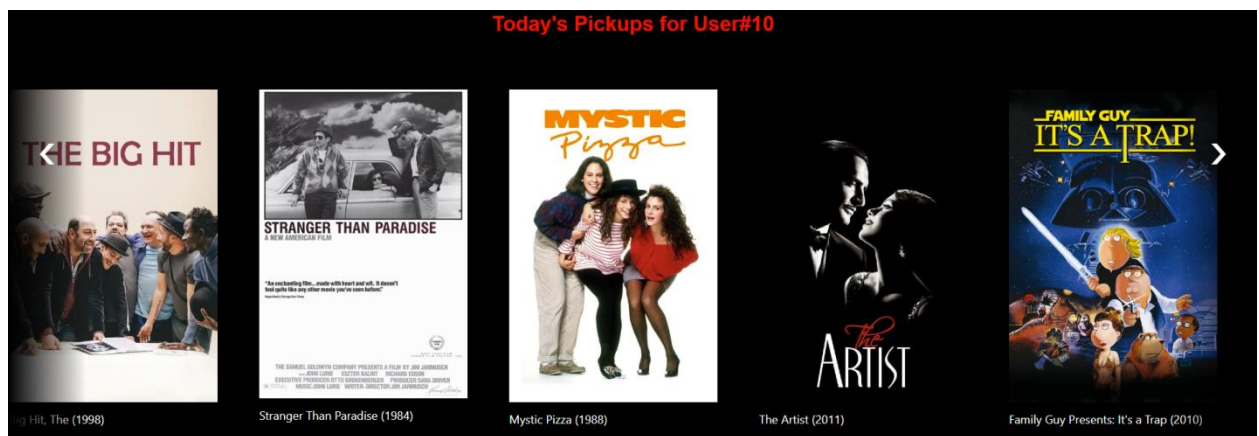


Figure 3.6 – Top 10 Recommendations for user #10 (0-5)

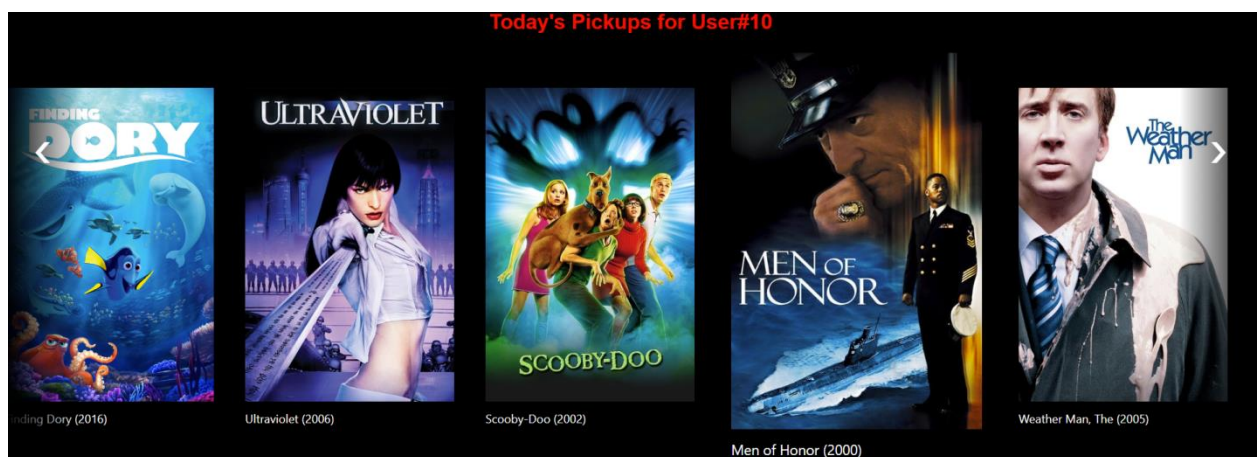


Figure 3.7 – Top 10 Recommendations for user #10 (5-10)

3.5.2 Backend

The backend, built with Flask, creates API endpoints to handle user requests. Apache Spark is used for processing user ratings data and training the ALS model. The model generates movie recommendations based on user input. Flask routes manage HTTP requests, and the recommendation results are sent back to the frontend. The ALS model's performance is evaluated using metrics like RMSE.

The TMDB API fetches movie posters to enhance the visual appeal of recommendations. Titles are formatted to ensure accurate API queries. Asynchronous requests fetch movie posters, with error handling ensuring efficiency.

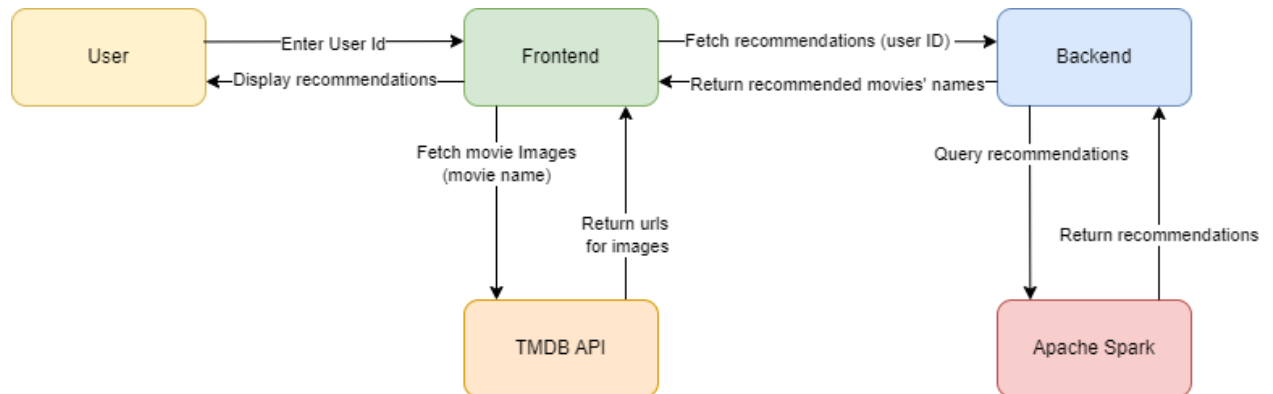


Figure 3.8 – Recommendation system diagram

User interactions in the CustomInput component trigger recommendation fetches from the backend. The backend processes the request and returns recommended movie titles, which the App component displays using the Carousel component. Loading states and error handling enhance user experience and reliability.

Conclusions to section 3

In this chapter, we successfully integrated a user interface with a backend recommendation engine to create a cohesive system that delivers personalized movie recommendations. The React frontend interacts seamlessly with the Flask backend, which processes user data and generates recommendations using the ALS algorithm implemented in Apache Spark.

Through this integration, we developed a dynamic and responsive user interface using React. The components handle user inputs and display movie recommendations efficiently, with React hooks ensuring effective state management and API integration. The Flask backend efficiently processes API requests, communicates with the Spark-based recommendation engine, and manages user ratings data to train the ALS model and generate recommendations. The backend's robustness and scalability allow it to handle multiple user requests simultaneously.

By integrating the TMDB API, we enhanced the user experience by fetching movie posters and additional details, making the recommendations visually appealing and informative. The interaction between the frontend and backend is smooth and efficient, enabling responsive and interactive user experiences as user inputs in the frontend trigger API calls to the backend, which processes the requests and returns the recommendations

The system's performance is evaluated using metrics such as RMSE, MAE, R-squared, Precision, and Recall, allowing us to optimize the model parameters and ensure accurate recommendations. Overall, the integration of the user interface with the backend recommendation engine results in a robust, efficient, and user-friendly recommendation system that provides accurate and personalized movie recommendations while enhancing the overall user experience.

4 FUNCTIONAL AND COST ANALYSIS OF A SOFTWARE PRODUCT

Functional Cost Analysis (FCA) is a methodology that enables the estimation of the true value of a product or service, independent of the company's organizational structure. FCA is conducted to identify cost savings by exploring more efficient production methods and achieving a better balance between the consumer value of a product and its manufacturing costs. This analysis utilizes economic, technical, and design information.

4.1 Formulation of the Design Problem

This paper applies the FCA method to conduct a feasibility analysis for developing a system that forecasts the sustainability of financial indicators. Since the design and implementation decisions of the system components impact the entire system, each individual subsystem must meet these standards. Therefore, the analysis focuses on the functions of a software product designed to collect, process, and analyze company data.

The technical requirements for the software product include:

- operation on personal computers with a standard set of components;
- user convenience and clarity;
- speedy data processing and real-time information access;
- easy scalability and maintenance;
- minimal implementation costs for the software product.

4.2 Justification of the Functions of the Software Product

Main Function F_0 : Develop a potential software product that enables the analysis of various characteristics directly affecting the sustainability of the enterprise. Based on this function, we can identify the following sub-functions:

- F_1 – the choice of the programs itself;
- F_2 – qualitative data analysis;
- F_3 – graphic all indicators.

Each of the main functions can be implemented in several ways:

Function F_1 : Python, JavaScript. Function F_2 : use of ready-made libraries, writing algorithms manually. Function F_3 : VS Code, Anaconda.

Based on the options presented, we build a morphological map (Figure 4.1).

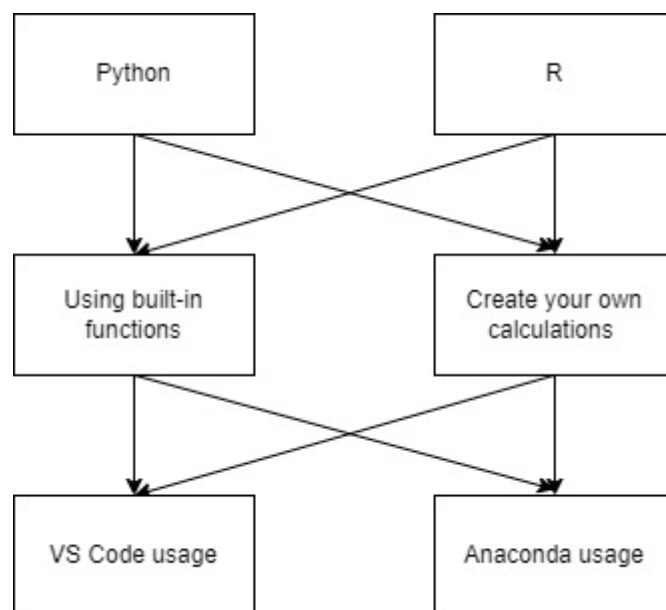


Figure 4.1 – Morphological map

The morphological map shows the set of all possible variants of the main functions. The positive-negative matrix is shown in Table 4.1.

Table 4.1 – Positive-negative matrix

Features.	Implementa tion options	Advantages.	Disadvantages.
F_1	A	Cross-platform, accessibility of libraries, takes less time when writing code	Dynamic typing,
	B	Easy to implement program for various calculations	It takes more time to development due to lack of libraries
F_2	A	Ease of implementation, saving time	Less flexibility
	B	Ideal for own products	The time spent, possible errors
F_3	A	Generally accepted realization	Sometimes does not meet to the expected values
	B	When performing their own research better able to communicate the findings	You need to spend a lot of a lot of time to write a program to build and find everything necessary in the task.

Based on the analysis of the positive-negative matrix, we conclude that certain function implementation options should be rejected during the development of the software product, as they do not align with the project's objectives. These unsuitable options are highlighted in the morphological map.

Function F_1 .

We prefer public accessibility. To simplify the work of writing code, option B should be rejected.

Function F_2 .

The program allows you to select both options. You can use options A/B.

Function F_3 .

Implementation of the first option is receptive to the program. This is option A.

Thus, we will consider the following option for the implementation of the PP: $F_1a - F_2a - F_3a, F_1a - F_2b - F_3a$.

To evaluate the quality of the functions under consideration, we chose the system of parameters described below.

4.3 Justification of the software product parameter system

Based on the data discussed above, the main selection parameters that will be used to calculate the technical level coefficient are determined. To describe the software product, we will use the following parameters:

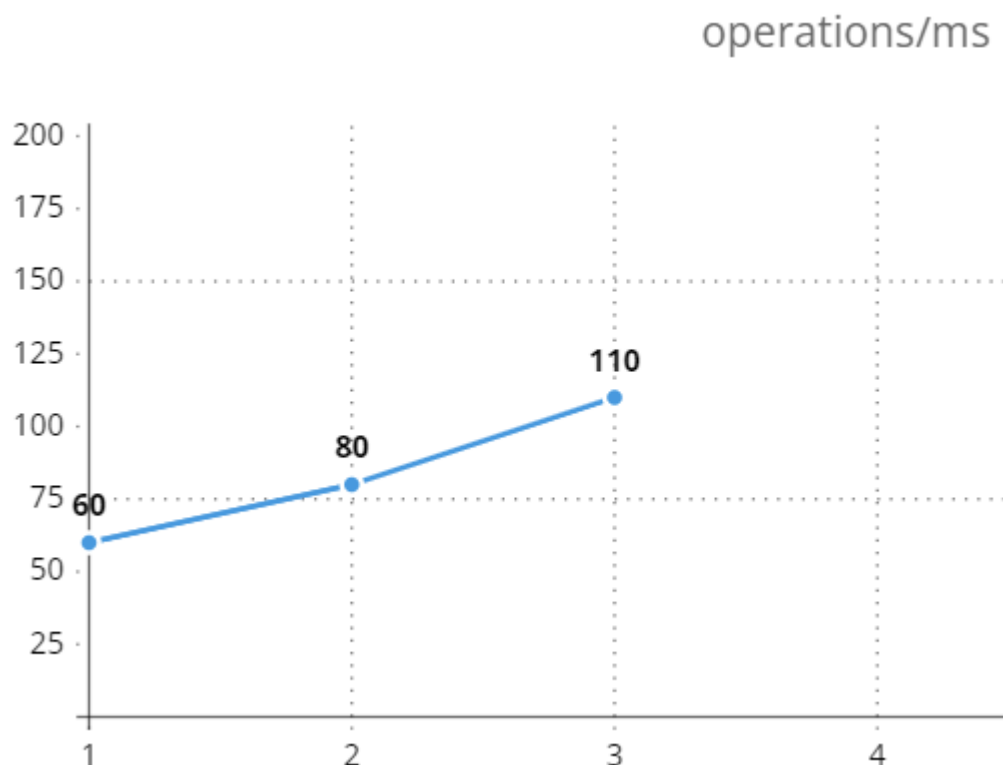
- X_1 is the speed of the programming language;
- X_2 is the amount of memory for computing and data storage;
- X_3 - data training time;
- X_4 is the potential amount of program code.

The worst, average and best values of parameters are selected based on customer requirements and conditions that characterize the operation of the software product, as shown in Table 4.3.

Table 4.3 – Parameters of characterization

Title. Parameter.	Symbols and notation	Units of measurement	The value of the parameter		
			worse	medium	best
The speed of the programming language	X_1	operations/ms	30	70	100
Memory capacity	X_2	Mb	65	45	30
Data pre-processing time	X_3	ms	80	50	20
Potential amount of software code	X_4	number of lines of code	350	250	200

According to the data of Table 4.3, graphical characteristics of the parameters are built – Figure 4.2 - Figure 4.5.

**Figure 4.2** – X_1 , the speed of the programming language

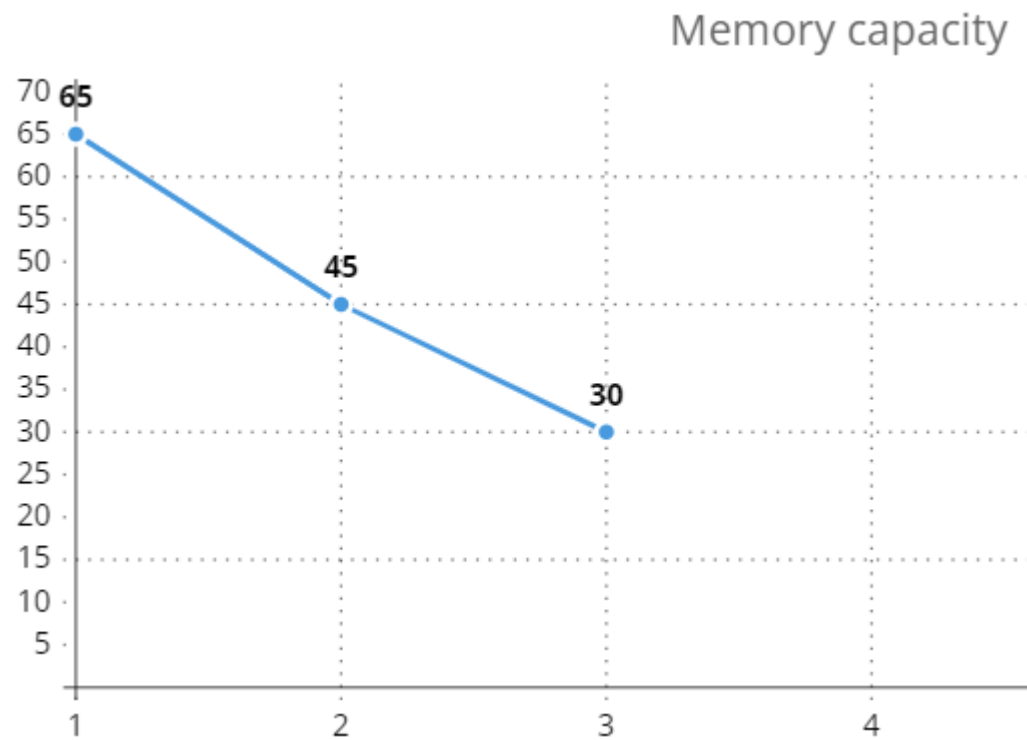


Figure 4.3 – X2, memory capacity

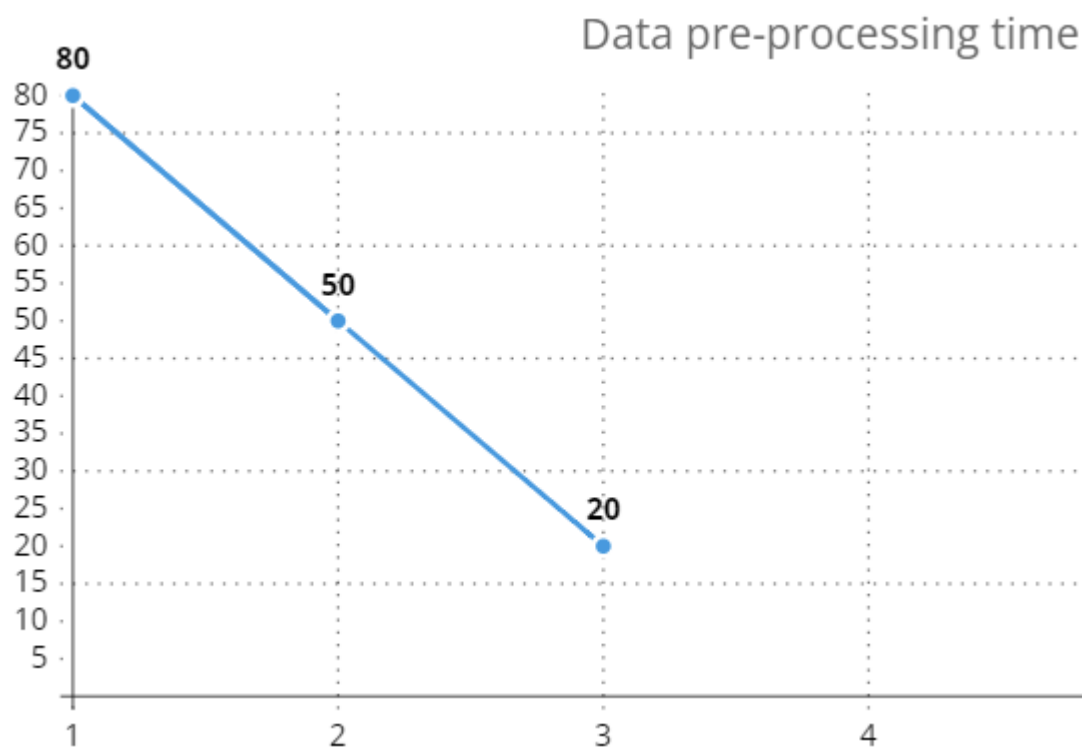


Figure 4.4 – X3, data pre-processing time

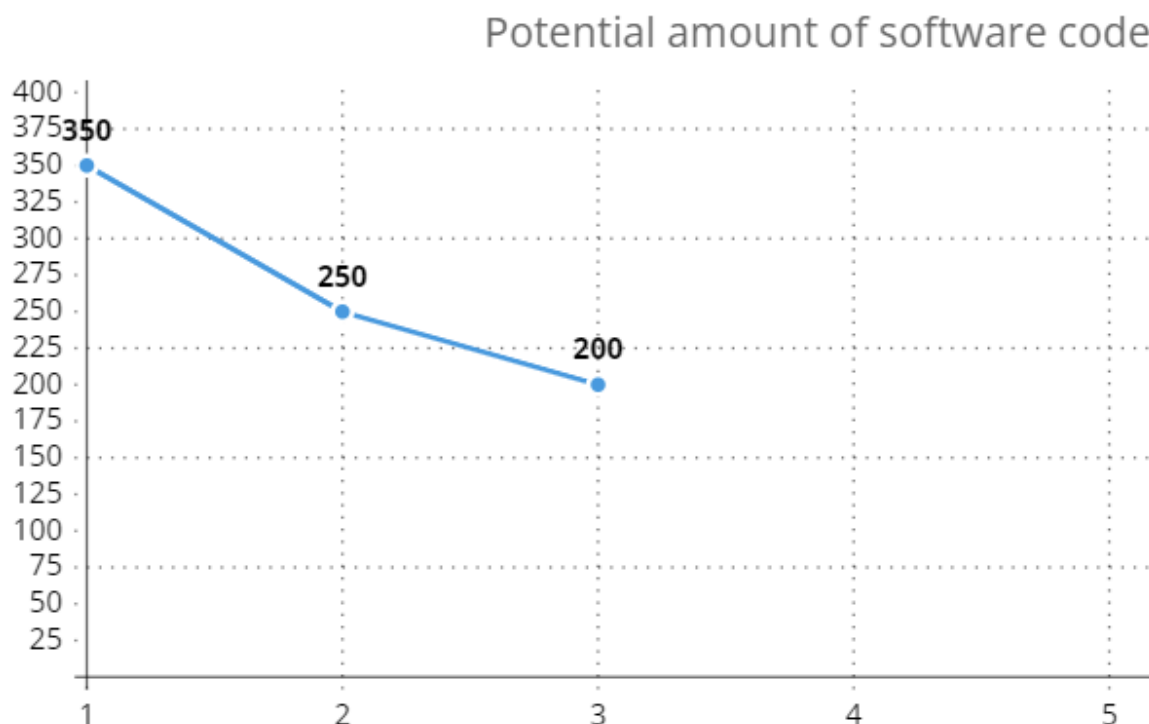


Figure 4.5 – X4, potential amount of software code

4.4 Analysing the expert evaluation of parameters

After thorough discussion and analysis, each expert evaluates the importance of each parameter for achieving the goal of developing a software product that provides the most accurate results in identifying parameters of adaptive forecasting models and calculating forecast values. The importance of each parameter is determined through pairwise comparison by an expert committee consisting of 7 members. The process of determining the significance coefficients includes:

- assigning different ranks to determine the level of significance of a parameter;
- checking the consistency of expert opinions for further use;
- establishing the pairwise priority of parameters;
- processing the results to determine the significance coefficient.

Table 4.3 – The results of the expert ranking

Parameter designation	Parameter name	Units of measurement	The rank of the parameter according to the expert's assessment							Sum of ranks R_1	Deviation Δ_1	Δ_1^2
			1	2	3	4	5	6	7			
X_1	The speed of the programming language	operations/ms	1	2	1	2	2	2	2	12	-5.5	30.25
X_2	Memory capacity	Mb	3	4	4	4	3	3	4	25	7.5	56.25
X_3	Data pre-processing time	ms	2	1	3	1	1	2	1	10	-7.5	56.25
X_4	Potential amount of software code	Number of lines of code	4	3	2	3	4	4	3	23	5.5	30.25
			10	10	10	10	10	10	10	70	0	173

To check the degree of reliability of expert opinions, we will define the following parameters.

The sum of the ranks of each of the parameters and the total sum of the ranks:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{N(n+1)}{2} = 70,$$

where N – number of experts;

n – number of parameters.

The average sum of ranks:

$$T = \frac{1}{n} R_{ij} = 17,5$$

Deviation of the sum of ranks of each parameter from the average sum of ranks:

$$\Delta_i = R_i - T.$$

The sum of deviations for parameters must be 0.

Total sum of squared deviations:

$$S = \sum_{i=1}^N \Delta_i^2 = 173.$$

Let's calculate the coefficient of consistency:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 173}{7^2(4^3 - 4)} = 0,706 > W_k = 0,67.$$

The ranking can be considered reliable, as the coefficient of consistency found exceeds the standard coefficient of 0.67.

Using the results of the ranking, we will conduct a pairwise comparison of all parameters and record the results in Table 4.4.

Table 4.4 – A pairwise comparison of all parameters and record the results

Parameters.	Experts							Final assessment	Numerical value
	1	2	3	4	5	6	7		
X1 and X2	<	<	<	<	<	<	<	<	0,5
X1 and X3	<	>	>	<	<	<	>	>	1,5
X1 and X4	<	<	<	<	<	<	<	<	0,5
X2 and X3	>	>	>	>	>	>	>	>	1,5
X2 and X4	<	>	<	<	>	<	<	>	1,5
X3 and X4	<	<	<	<	<	<	<	<	0,5

The numerical value that determines the degree of superiority of the i -th parameter over the j -th parameter, a_{ij} , is determined by the formula:

$$a_{ij} = \{1.5 \text{ for } X_i > X_j \text{ } 1.0 \text{ for } X_i = X_j \text{ } 0.5 \text{ for } X_i < X_j\}$$

From the obtained numerical estimates of preference, we will compile the matrix $A = \parallel a_{ij} \parallel$.

For each parameter, we calculate the weight of K_{wi} using the following formulas:

$$K_{wi} = \frac{b_i}{\sum_{i=1}^n b_i}$$

$$b_i = \sum_{j=1}^N a_{ij}$$

The relative scores are calculated several times until the next values differ slightly from the previous ones (less than 2%). In the second and subsequent steps, the relative scores are calculated using the following formulas:

$$K_{Bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j$$

Table 4.5 – Calculation of parameter weights

Parameters. $x_{\$}$	Parameters. $x_{\%}$				The first iteration.		The second iteration.		The third iter	
	X1	X2	X3	X4	$b_{\$}$	K_{Bi}	$b_{\$}$!	K_{Bi} !	$b_{\$}$ "	K_{Bi} "
X1	1	0,5	1,5	0,5	3,5	0,22	12,25	0,21	44,875	0,21
X2	1,5	1	1,5	1,5	5,5	0,34	21,25	0,36	77,875	0,36
X3	0,5	0,5	1	0,5	2,5	0,16	9,25	0,16	34,125	0,15
X4	1,5	0,5	1,5	1	4,5	0,28	16,25	0,27	59,125	0,28
Total:					16	1	59	1	216	1

As can be seen from Table 4.5, the difference in the values of the weighting coefficients does not exceed 2%, so no more iterations are required.

4.5 Analysis of the quality level of options for implementing functions

We determine the quality level of each variant of the main functions separately.

The absolute values of the parameters $X2$ (memory capacity), $X3$ (data preprocessing time) and $X4$ (potential program code size) meet the technical requirements of the functioning conditions of this software package.

The absolute value of the parameter *X1* (the speed of the programming language) was not the worst.

The technical level coefficient for each implementation option is calculated as follows (Table 4.6):

$$K_K(j) = \sum_{i=1}^n K_{wi,j} B_{i,j},$$

where *n* - number of parameters;

K_{wi} – is the weighting factor of the *i-th* parameter;

B_i – the score of the *i-th* parameter in points.

Table 4.6 – Calculation of quality level indicators of options for implementing the main functions of the PP

Main functions	Option for implementing the function	Parameters.	Absolute value of the parameter	Scoring of the parameter	Parameter weighting factor	Quality level coefficient
F1	A	X1	98	23	0,16	3,68
F3	A	X2	85	27	0,26	7,02
	B	X3	25	17	0,2	3,4
F4	A	X4	23	21	0,38	7,98

Based on data from Table above using the formula:

$$K_K = K_{QC}[F_{1k}] + K_{TY}[F_{2k}] + \dots + K_{QC}[F_{zk}],$$

determine the quality level of each option:

$$K_{K1} = 2,52 + 5,4 + 1,2 = 9,12;$$

$$K_{K2} = 2,52 + 5,4 + 2,52 = 10,44$$

As can be seen from the calculations, the preferred option is option 2, for which the technical level coefficient is the highest.

4.6 Economic analysis of the options for developing the PP

To determine the cost of developing a software application, let's first calculate the labour intensity.

All options include two separate tasks.

1. Development of a software product project.
2. Software development.

Task 1 belongs to Group A in terms of novelty, Task 2 - to Group B. In terms of complexity, the algorithms used in Task 1 belong to Group 1; and in Task 2 - to Group 3.

Task 1 uses background information, and task 2 uses information in the form of data.

We will calculate the time standards for development and programming for each of the tasks. The total labour intensity is calculated as:

$$T_D = T_P \cdot K_C \cdot K_{CC} \cdot K_M \cdot K_{ST} \cdot K_{ST.M},$$

where T_D is the labour intensity of the PP development;

K_C - correction coefficient;

K_{CC} is the coefficient for the complexity of the input information;

K_M is the coefficient of the programming language level;

K_{ST} is the ratio of the use of standard modules and applications;

$K_{ST.M}$ - coefficient of standard mathematical support.

For the first task, based on the time norms for computational tasks of novelty degree A and complexity group 1 of the algorithm, the labour intensity is equal to: $T_D = 60$ person-days. Correction factor that takes into account the type of regulatory and reference information for the first task: $K_C = 1.7$. The correction factor that takes into account the complexity of controlling incoming and outgoing information for all seven tasks is 1: $K_{CC} = 1$. Since standard modules are used in the development of the first task, we will take this into account with the coefficient $K_{ST} = 0.8$. Then the total complexity of programming the first task is equal:

$$T_1 = 37 \cdot 1.7 \cdot 0.8 = 81,6 \text{ man-days.}$$

Let's make similar calculations for further tasks. For the second task (using the algorithm of the third complexity group, degree of novelty B), i.e. $T_D = 32$ man-days, $K_C = 0.9$, $K_{CC} = 1$, $K_{ST} = 0.8$:

$$T_2 = 32 \cdot 0.9 \cdot 0.8 = 23.04 \text{ man-days.}$$

We add up the labour intensity of the relevant tasks for each of the selected programme implementation options to get their labour intensity:

$$T_I = (81.6 + 23.04 + 4.8 + 23.04) \cdot 8 = 1059.84 \text{ man-hours.}$$

$$T_{II} = (81.6 + 23.04 + 6.91 + 23.04) \cdot 8 = 1076.72 \text{ man-hours.}$$

Option II has the highest labour intensity. Two programmers with a salary of UAH 36600 and one data analyst with a salary of UAH 25000 are involved in the development. Determine the average salary per hour using the formula:

$$AS = \frac{M}{T_m \cdot t} UAH,$$

where M is the monthly salary of employees;

T_m – number of working days per week;

t – the number of working hours per day.

$$AS = \frac{36600 + 36600 + 25000}{3 \cdot 21 \cdot 8} = 194.84 \text{ UAH}$$

Then, let's calculate the average paid salary using the formula:

$$APS = W_h \cdot L_i \cdot W_a,$$

where W_h is the hourly wage of a programmer;

L_i – the labour intensity of the relevant task;

W_a is a standard that takes into account additional wages.

The salary of developers by option is:

$$\text{I. } W_d = 194.84 \cdot 1059.84 \cdot 1.2 = 247799.07 \text{ UAH.}$$

$$\text{II. } W_d = 194.84 \cdot 1076.72 \cdot 1.2 = 251745.75 \text{ UAH.}$$

The unified social tax contribution is 22%:

$$\text{I. } W_{def} = 247799.07 \cdot 0.22 = 54515.79 \text{ UAH.}$$

$$\text{II. } W_{def} = 251745.75 \cdot 0.22 = 55384.07 \text{ UAH}$$

Now let's determine the cost of paying for one machine hour. W_m

Since one computer serves one programmer with a salary of 36600 UAH, with an employment rate of 0.2, we get the following for one machine:

$$W_d = 12 * M * K_s * 0.2 = 87840 \text{ UAH}$$

Including additional salary:

$$C = W_d(1 + 0.2) = 105408 \text{ UAH}$$

Social security contributions:

$$C_{SSC} = C \times 0.22 = 105408 * 0.22 = 23189.76 \text{ UAH}$$

Depreciation charges are calculated at a depreciation rate of 25% and a computer cost of UAH 30,000.

$$C_{Dep} = K_{TM} \times K_A \times C_{Contract} = 1.15 \times 0.25 \times 30000 = 8625 \text{ UAH,}$$

where K_{TM} is a coefficient that takes into account the cost of transport and installation of the device at the user's premises;

K_A – annual depreciation rate;

$C_{Contract}$ – the contract price of the device.

We calculate repair and maintenance costs as follows:

$$C_{\text{Repair}} = K_{\text{TM}} \times C_{\text{Contract}} \times K_{\text{Repair}} = 1.15 \times 30000 \times 0.05 = 1725 \text{ UAH}$$

where K_{Repair} is the percentage of expenditures on current repairs.

We calculate the effective hourly time fund of a PC for a year using the formula:

$$T_{\text{Eff}} = (D_{\text{Cal}} - D_{\text{Off}} - D_{\text{Holidays}} - D_{\text{Repair}}) \times t_{\text{day}} \times K_{\text{Util}} = (365 - 104 - 12 - 16) \times 8 \times 0.9 = 1677.6 \text{ hours},$$

where D_{Cal} is the calendar number of days in a year;

D_{Off} , D_{Holidays} – the number of days off and holidays, respectively;

D_{Repair} – number of days of scheduled equipment repairs;

t_{day} is the number of working hours per day;

K_{Util} is the device utilisation rate over time during a shift.

We calculate electricity costs using the formula:

$$C_{\text{Electricity}} = T_{\text{Eff}} \times N_{\text{Power}} \times K_s \times C_{\text{Tariff}} = 1324.8 \times 0.7 \times 0.2 \times 5.23 = 970.019 \text{ UAH},$$

where N_{Power} is the average power consumption of the device;

K_s is the device occupancy rate;

C_{Tariff} – tariff per 1 kWh of electricity.

Overheads are calculated using the formula:

$$C_{\text{Overhead}} = C_{\text{Contract}} \times 0.67 = 30000 \times 0.67 = 20100 \text{ UAH}$$

Then, the annual operating costs will be lower:

$$\begin{aligned} C_{\text{Operating}} &= C + C_{\text{SSC}} + C_{\text{Dep}} + C_{\text{Repair}} + C_{\text{Electricity}} + C_{\text{Overhead}} \\ &= 105408 + 23189,76 + 8625 + 1725 + 970,019 + 20100 \\ &= 160017,78 \text{ UAH} \end{aligned}$$

The cost of one machine hour of a computer will be equal:

$$C_{\text{MH}} = \frac{C_{\text{Operating}}}{T_{\text{Eff}}} = \frac{160017,78}{1677,6} = 95.38 \text{ UAH/hour}$$

Since in this case all the work related to the development of a software product is carried out on a computer, the cost of paying for machine time, depending on the chosen implementation option, is:

- I. $C_{\text{M}} = C_{\text{MH}} \times T = 95,38 \cdot 1059,84 = 101087,54 \text{ UAH}$
- II. $C_{\text{M}} = 95,38 \cdot 1076,72 = 102697,55 \text{ UAH}$

Overheads account for 67% of salaries:

- I. $C_{\text{Overhead}} = C_{3\Pi} \times 0.67 = 247799,07 \cdot 0,67 = 166025,38 \text{ UAH}$
- II. $C_{\text{Overhead}} = 251745,75 \cdot 0,67 = 168669,65 \text{ UAH}$

Thus, the cost of developing the PP by option is as follows:

$$C_{Dev} = C_{3\Pi} + C_{SSC} + C_M + C_{Overhead}$$

- I. $C_{Dev} = 247799,07 + 54515,79 + 101087,54 + 166025,38 = 569427,11 \text{ UAH}$
- II. $C_{Dev} = 251745,75 + 55384,07 + 102697,55 + 168669,65 = 578497,02 \text{ UAH}$

4.7 Selection of the best option for the PP at the technical and economic level

Let's calculate the technical and economic level coefficient using the formula:

$$K_{TechEco} = \frac{K_{Coeff}}{C_{Dev}}$$

$$K_{TechEco1} = \frac{9,12}{569427,11} = 1,602 \times 10^{-5}$$

$$K_{TechEco2} = \frac{10,44}{578497,02} = 1,805 \times 10^{-5}$$

As we can see, the most effective is the first option of the program implementation with the technical and economic level coefficient $K_{TEP1} = 1,805 \cdot 10^{-5}$.

After performing a functional and cost analysis of the software complex under development, it can be concluded that of the alternatives remaining after the first selection of two variants of the software complex, the first variant of the software product implementation is optimal. It has the best indicator of the technical and economic level of quality

$$K_{\text{TechEco1}} = 1,805 \cdot 10^{-5}$$

This option of the software product has the following parameters.

1. The choice of a software product is Python.
2. Implement an important production with the help of built-in functions.
3. Environment for development – VS Code.

This option of the software package provides the user with a user-friendly interface, fast implementation of the program and accessible functionality for work.

Conclusions to section 4

In this section, a comprehensive functional and cost analysis of the software product was conducted, along with an assessment of its primary functions. The study presents various implementation options to ensure the most accurate and optimal selection strategy, considering economic factors and compatibility with the future software product.

As a result of the functional and cost analysis, the main functions of the software product were identified and evaluated, and the parameters characterizing it were determined. Based on this analysis, an implementation option for the software product was selected.

SUMMARY

The exponential growth of streaming platforms like Netflix, Spotify, and YouTube has revolutionized media consumption, necessitating intelligent recommender systems to enhance user satisfaction. This research aimed to deepen knowledge in data mining methods to solve content recommendation problems, focusing on analyzing, developing, and implementing effective recommender systems.

Key findings include the evolution and importance of recommender systems, highlighting their role in modern platforms. Content-based and collaborative filtering approaches were explored, with hybrid systems combining multiple strategies for better performance. Challenges such as the cold-start problem, sparsity, scalability, shilling attacks, and the gray sheep problem were addressed.

Data collection utilized explicit and implicit user interactions, preparing data for collaborative filtering with Apache Spark. The Alternating Least Squares (ALS) algorithm was chosen for its effectiveness, evaluated using metrics like RMSE, MAE, and precision. Integration with a user-friendly interface developed using React, Flask, and TMDb API enhanced the user experience.

A comprehensive functional and cost analysis identified and evaluated the main functions, leading to the selection of an implementation option using Python, built-in functions, and VS Code. This approach provides a robust, efficient, and user-friendly recommendation system, offering accurate and engaging content recommendations while enhancing overall user satisfaction.

REFERENCES

1. Huttner J. From Tapestry to SVD: A survey of the algorithms that power recommender systems; 2009.
2. Sharma, Richa & Singh, Rahul. (2016). Evolution of Recommender Systems from Ancient Times to Modern Era: A Survey. Indian Journal of Science and Technology. 9. 10.17485/ijst/2016/v9i20/88005.
3. Chong, D. (2020, April 30). Deep dive into Netflix's recommender system: How Netflix achieved 80% stream time through personalization. Towards Data Science. URL: <https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48>
4. Bartlett, J. (2024, March 10). Recommendation algorithms as therapist: What are the machines trying to tell you? URL: <https://jamiejbartlett.substack.com/p/recommendation-algorithms-as-therapist>
5. Mohsin, M. (2023, August 12). 10 TikTok statistics that you need to know in 2023 [Infographic]. Oberlo. URL: <https://www.oberlo.com/blog/tiktok-statistics>
6. Statista. (2022). Hours of video uploaded to YouTube every minute as of February 2022. URL: <https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/>
7. Implementing Recommender Systems using Machine Learning and Knowledge Discovery Tools - Scientific Figure on ResearchGate. URL: https://www.researchgate.net/figure/The-main-types-of-recommendation-system_fig1_354259927
8. Chaturvedi, Akshay & Peleja, Filipa & Freire, Ana. (2017). Recommender System for News Articles using Supervised Learning.
9. Murel, J., & Kavlakoglu, E. (2024, March 21). Content-based filtering. IBM. URL: <https://www.ibm.com/topics/content-based-filtering#f01>

10. Roy, A. (2020, July 29). Introduction to recommender systems- 1: Content-based filtering and collaborative filtering. Towards Data Science. URL: <https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421>
11. Nguyen, H. T. V. (2020, March 17). Entwicklung eines Recommender Systems: Unterschiedliche Strategien anhand eines Travel- & Outdoor Online Shops.
12. Zhang, R.-s., Liu, Q.-d., Chun-Gui, Wei, J.-X., & Ma, H. (2014). Collaborative filtering for recommender systems. In Proceedings of the 2014 Second International Conference on Advanced Cloud and Big Data (pp. 301-307). IEEE
13. Kundegraber, K., & Pletzl, S. (2022, December 4). Basic approaches in recommendation systems. Graz University of Technology
14. Ekstrand, Michael D. & Riedl, John T. & Konstan, Joseph A.: „Collaborative Filtering Recommender Systems“, Foundations and Trends in Human Computer Interaction, Vol. 4, No. 2, S.81-173, Ohne Ort, Februar 2011
15. F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5,4: 19:1–19:19. URL: <https://doi.org/10.1145/2827872>
16. Delua, J. (n.d.). Supervised versus unsupervised learning: What's the difference? IBM Analytics, Data Science/Machine Learning. URL: <https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning>
17. Benjamin Wang. "Ranking Evaluation Metrics for Recommender Systems: Various evaluation metrics are used for evaluating the effectiveness of a recommender." Towards Data Science, Jan 18, 2021. URL: <https://towardsdatascience.com/ranking-evaluation-metrics-for-recommender-systems-263d0a66ef54>.
18. Cha, G.-W., Moon, H., Kim, Y.-M., Hong, W.-H., Hwang, J.-H., Park, W.-J., & Kim, Y.-C. (2020). Development of a prediction model for demolition waste generation using a random forest algorithm based on small datasets.

- International Journal of Environmental Research and Public Health, 17(19), 6997. <https://doi.org/10.3390/ijerph17196997>
19. Flanagan, D. (2020). JavaScript: The Definitive Guide. O'Reilly Media.
 20. Banks, A., & Porcello, E. (2020). Learning React: Functional Web Development with React and Redux. O'Reilly Media.
 21. Beazley, D., & Jones, B. K. (2013). Python Cookbook. O'Reilly Media.
 22. Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
 23. Hug, N. (2017). Surprise: A Python library for recommender systems. Retrieved from <http://surpriselib.com/>
 24. Zaharia, M., et al. (2016). Apache Spark: Unified Analytics Engine for Big Data. Retrieved from <https://spark.apache.org/>
 25. Xin, R. S., et al. (2015). Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM, 59(11), 56-65.
 26. Microsoft. (n.d.). Visual Studio Code Documentation. URL: <https://code.visualstudio.com/docs>
 27. Chiang, J. (2021, May 16). Model-based Recommendation System with Matrix Factorization — ALS Model and The Math behind. Analytics Vidhya.

APPENDIX A – PROGRAMCODE

```

App.py
from flask import Flask, request, jsonify
from flask_cors import CORS
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
from MovieLens import MovieLens

app = Flask(__name__)
CORS(app)
# Initialize Spark session and model
spark = SparkSession.builder.appName("ALSExample").getOrCreate()

lines = spark.read.option("header", "true").csv("./ml-latest-small/ratings.csv").rdd
ratingsRDD = lines.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=float(p[2]), timestamp=int(p[3])))
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])

als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId",
ratingCol="rating",
        coldStartStrategy="drop")
model = als.fit(training)

predictions = model.transform(test)

```

```

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

ml = MovieLens()
ml.loadMovieLensLatestSmall()

def get_recommendations_for_user(user_id, num_recommendations=10):
    user_df = spark.createDataFrame([(user_id,)], ["userId"])
    user_recs = model.recommendForUserSubset(user_df,
num_recommendations)
    return user_recs.collect()

@app.route('/recommend', methods=['GET'])
def recommend():
    user_id = int(request.args.get('userId'))
    num_recommendations = int(request.args.get('num', 10))

    user_recommendations = get_recommendations_for_user(user_id,
num_recommendations)

    recommendations = []
    for row in user_recommendations:
        for rec in row.recommendations:
            recommendations.append(ml.getMovieName(rec.movieId))

    return jsonify(recommendations)

if __name__ == "__main__":

```

```
app.run(host='0.0.0.0', port=5000)
spark.stop()
```

MovieLens.py

```
import os
import csv
import re
from collections import defaultdict
from surprise import Dataset, Reader

class MovieLens:

    def __init__(self):
        self.movieID_to_name = {}
        self.name_to_movieID = {}
        self.ratingsPath = './ml-latest-small/ratings.csv'
        self.moviesPath = './ml-latest-small/movies.csv'

    def loadMovieLensData(self):
        self.movieID_to_name.clear()
        self.name_to_movieID.clear()

        reader = Reader(line_format='user item rating timestamp', sep=',',
skip_lines=1)
        ratingsDataset = Dataset.load_from_file(self.ratingsPath, reader=reader)

        with open(self.moviesPath, newline="", encoding='ISO-8859-1') as csvfile:
            movieReader = csv.reader(csvfile)
            next(movieReader) # Skip header line
            for row in movieReader:
```

```

        movieID = int(row[0])
        movieName = row[1]
        self.movieID_to_name[movieID] = movieName
        self.name_to_movieID[movieName] = movieID

    return ratingsDataset

def fetchUserRatings(self, userID):
    userRatings = []
    foundUser = False
    with open(self.ratingsPath, newline='') as csvfile:
        ratingReader = csv.reader(csvfile)
        next(ratingReader)
        for row in ratingReader:
            currentUserID = int(row[0])
            if currentUserID == userID:
                movieID = int(row[1])
                rating = float(row[2])
                userRatings.append((movieID, rating))
                foundUser = True
            elif foundUser:
                break
    return userRatings

def calculatePopularityRanks(self):
    ratingCounts = defaultdict(int)
    rankings = defaultdict(int)
    with open(self.ratingsPath, newline='') as csvfile:
        ratingReader = csv.reader(csvfile)
        next(ratingReader)

```

```

        for row in ratingReader:
            movieID = int(row[1])
            ratingCounts[movieID] += 1
    rank = 1
    for movieID, count in sorted(ratingCounts.items(), key=lambda x: x[1],
reverse=True):
        rankings[movieID] = rank
        rank += 1
    return rankings

def extractGenres(self):
    genres = defaultdict(list)
    genreIDMap = {}
    maxGenreID = 0
    with open(self.moviesPath, newline="", encoding='ISO-8859-1') as csvfile:
        movieReader = csv.reader(csvfile)
        next(movieReader)
        for row in movieReader:
            movieID = int(row[0])
            genreList = row[2].split('|')
            genreIDList = []
            for genre in genreList:
                if genre not in genreIDMap:
                    genreIDMap[genre] = maxGenreID
                    maxGenreID += 1
                genreIDList.append(genreIDMap[genre])
            genres[movieID] = genreIDList

    for movieID, genreIDList in genres.items():
        bitfield = [0] * maxGenreID

```

```

for genreID in genreIDList:
    bitfield[genreID] = 1
genres[movieID] = bitfield

```

```

return genres

```

```

def extractYears(self):
    yearPattern = re.compile(r"(?:\((\d{4})\))?\s*$")
    years = defaultdict(int)
    with open(self.moviesPath, newline="", encoding='ISO-8859-1') as csvfile:
        movieReader = csv.reader(csvfile)
        next(movieReader)
        for row in movieReader:
            movieID = int(row[0])
            title = row[1]
            match = yearPattern.search(title)
            year = match.group(1) if match else None
            if year:
                years[movieID] = int(year)
    return years

```

```

def fetchMiseEnSceneFeatures(self):
    miseEnScene = defaultdict(list)
    with open("LLVisualFeatures13K_Log.csv", newline="") as csvfile:
        mesReader = csv.reader(csvfile)
        next(mesReader)
        for row in mesReader:
            movieID = int(row[0])
            features = [float(value) for value in row[1:8]]
            miseEnScene[movieID] = features

```

```
return miseEnScene
```

```
def getMovieName(self, movieID):
    return self.movieID_to_name.get(movieID, "")

def getMovieID(self, movieName):
    return self.name_to_movieID.get(movieName, 0)
```

Frontend:

App.js

```
import { useState, useEffect } from "react";
import "./App.scss";
import Carousel from "./components/Carousel.jsx";
import CustomInput from "./components/CustomInput.jsx";

function App() {
    const [items, setItems] = useState([]);
    const [isLoading, setIsLoading] = useState(true);
    const [id, setId] = useState(1);

    useEffect(() => {
        setIsLoading(true);
        fetch(`http://localhost:5000/recommend?userId=${id}&num=10`)
            .then((response) => response.json())
            .then((data) => setItems(data));
        setIsLoading(false);
    }, [id]);

    if (isLoading) return <h1>Loading...</h1>;

    return (
```

```

    <div className="App">
      <CustomInput onChange={setId} />
      <Carousel items={items} id={id} />
    </div>
  );
}

```

```
export default App;
```

Carousel.jsx

```
import React, { useState, useEffect } from "react";
```

```
import "../styles/Carousel.scss";
```

```

function formatTitle(title) {
  const titleWithoutYear = title.replace(/\s*(\d{4})$/, "");
  const formattedTitle = titleWithoutYear.replace(/\s+/g, "+");
  return formattedTitle;
}

```

```

async function fetchMoviePoster(movieName) {
  const url =
    `https://api.themoviedb.org/3/search/movie?query=${movieName}&api_key=78b488c1a896981b9898189410f98a9d`;

```

```

  try {
    const response = await fetch(url);
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const movieData = await response.json();

```

```

    const posterPath = movieData.results[0].poster_path;
    const posterUrl = `https://image.tmdb.org/t/p/w500/${posterPath}`;
    return posterUrl;
  } catch (error) {
    console.error("Error fetching movie poster:", error);
    return null;
  }
}

```

```

const Carousel = ({ items, id }) => {
  const [movies, setMovies] = useState([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    setIsLoading(true);

    const fetchMovies = async () => {
      const moviesWithPosters = await Promise.all(
        items.map(async (item) => {
          const title = formatTitle(item);
          const posterUrl = await fetchMoviePoster(title);
          console.log(item);
          return { title: item, posterUrl };
        })
      );

      setMovies(moviesWithPosters);
      setIsLoading(false);
    };
  });
}

```

```

    fetchMovies();
  }, [items]);

  if (isLoading || !items) return <div>Loading...</div>;

  return (
    <>
      <h1>Today's Pickups for User{`#{id}`}</h1>
      <div className="wrapper">
        <section id="section1">
          <a href="#section2" className="arrow__btn">
            <
          </a>
          {movies.slice(0, 5).map((movie, index) => {
            return (
              <div key={index} className="item">
                <img src={movie.posterUrl} alt={movie.title} />
                <p>{movie.title}</p>
              </div>
            );
          })}
          <a href="#section2" className="arrow__btn">
            >
          </a>
        </section>
        <section id="section2">
          <a href="#section1" className="arrow__btn">
            <
          </a>
          {movies.slice(5, 10).map((movie, index) => {

```

```

    return (
      <div key={index} className="item">
        <img src={movie.posterUrl} alt={movie.title} />
        <p>{movie.title}</p>
      </div>
    );
  })}
  <a href="#section1" className="arrow__btn">
    ›
  </a>
</section>
</div>
</>
);
};

```

```
export default Carousel;
```

CustomInput.jsx

```
import {useState} from 'react'
```

```

const CustomInput = ({onChange}) => {
  const [currentValue, setCurrentValue] = useState("");

```

```

  const handleChange = (e) => {
    setCurrentValue(e.target.value);
  }

```

```

const handleClick = () => {
  onChange(currentValue);
  setCurrentValue("");
}

return (
  <div className='input-container'>
    <input placeholder="type user id" className='user-input' type="number"
value={currentValue} onChange={handleChange} />
    <button                                className='submit-button'
onClick={handleClick}>Reccomend</button>
  </div>
)
}

export default CustomInput

```